Integrating Pretrained Large Language Models and Graph Neural Networks For Conceptual Modeling: A Graph Language Modeling Framework

IEEE Publication Technology, *Staff, IEEE,*

***Abstract*—Software systems are becoming increasingly complex, making effective and efficient conceptual modeling essential for understanding and communicating system designs. Conceptual models (CMs) use natural language terms, modeling language syntax, and ontological structures to embed rich textual and graph-based conceptual modeling patterns within a model. These patterns can be learned from large CM datasets to support model generation, modification, and refactoring tasks. While Large Language Models (LLMs) and Graph Neural Networks (GNNs) have emerged as effective ML techniques for capturing patterns within text and graph data, respectively, existing approaches often (i) under-utilize LLMs' language comprehension capabilities, (ii) rely solely on GNNs for utilizing structural information, or (iii) fail to integrate both modalities. To mitigate these limitations, we introduce GLM4MDE, a framework that combines fine-tuned LLM embeddings with GNNs to produce semantically and structurally enriched model representations for modeling assistance. We describe its architecture, training procedures, and integration strategies, and then evaluate it on multiple CM datasets from heterogeneous modeling languages. We provide benchmarks for several CM-related tasks. Furthermore, we provide ablation studies that quantify the impact of key configuration parameters in training the ML model on the model datasets. Finally, to enable adoption, we provide an open-source Python library with a modular API, documentation, and example notebooks for applying GLM4MDE across diverse modeling environments.**

***Index Terms*—Machine Learning, Graph Neural Networks, Pretrained Language Models, Conceptual Modeling, Model-Driven Engineering, Software Engineering**

I. Introduction

Conceptual models explicitly represent descriptive and prescriptive domain knowledge. In enterprise and information systems engineering, a domain may include—but is not limited to—business processes, information structures, transactions, and value exchanges [1]. Model-driven engineering (MDE) is a development paradigm that treats these conceptual models[1] as the central artifacts throughout the software lifecycle. Defined via graphical notations, these models can be systematically transformed and refined to produce executable code, configuration files, documentation, and other deliverables [2]. Current MDE practices depend on domain-specific languages (DSLs), general-purpose modeling languages (e.g., UML, SysML), and ontology-based formalisms (OntoUML), which introduce interoperability barriers between different modeling languages and steep learning curves for practitioners [2]. A recent cross-disciplinary survey on software, data, and process modeling highlighted that modeling environments must be easy to use, see wider practitioner adoption, and offer greater automation, since building models from scratch is both time-and effort-intensive [3]. Existing publicly accessible repositories, especially the datasets that embrace the FAIR principles [4], [5], can enable modeling patterns reuse, adaptation, and learning, thereby fostering vibrant modeling communities and empirical data-driven research [6]. However, these collective resources remain under-exploited without robust techniques to extract, represent, and leverage the rich semantics encoded in conceptual models.

Concurrently, AI has emerged as a promising avenue to enhance MDE: publications in ML for MDE (ML4MDE) have tripled over the past five years [7]. Techniques from Deep Learning and Natural Language Processing have been applied to Partial Model Completion (PMC) [8]–[10], automated domain model extraction [11], [12], model transformation [13], [14], and metamodel classification and clustering (MCC) [15], [16]. Despite this growing recognition of AI's role in conceptual modeling [3], there are several limitations with ML4MDE research starting from data encoding, utilization, and ML models training within the ML4MDE pipeline; therefore, in this work, we aim to clearly understand the data in conceptual models to design effective ML solutions.

ML solutions for conceptual modeling first encode the conceptual model's constituent information in a representation suitable for training an ML model, i.e., a numerical matrix. Then, the ML model is trained to learn the knowledge encoded in the models. Finally, the trained ML models are used for modeling tasks like model transformation, PMC, or MCC. It is important to note that in order for an ML model to predict a missing element or to classify a model into a particular domain, the ML model should have (learned) knowledge about the constituting semantics encoded within a model.

The contextual information that captures representative semantics of a model needs to be accessible to the ML model during training for the model to *learn* semantically rich patterns. Fig. 1 shows four sources of semantics involved in a model rooted in the lexical terms and the graph structure of the model. The lexical terms represent informal natural language semantics and formal semantics like modeling language semantics, i.e., modeling language primitives (e.g., classes,

This paper was produced by the IEEE Publication Technology Group. They are in Piscataway, NJ.

Manuscript received April 19, 2021; revised August 16, 2021.

[1]Note that throughout the paper, we will use the term 'model' to relate to a conceptual model, whereas ML models will be referred to as 'ML model,' to omit confusion.

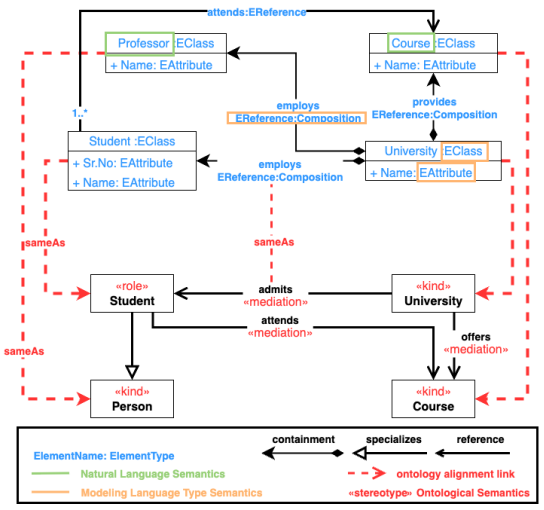


Fig. 1. Sources of Semantics in a model

relations, attributes) and ontological semantics encoded in the model's elements, coming from foundational ontologies like UFO [17]. Fig. 1 shows that the term "Professor" is not defined anywhere and is used as an English language term during modeling a University model, whereas the term "EClass", "EReference" are defined by the Ecore metamodel, and "kind", "role" are terms defined by UFO. Moreover, given that a model is a graph, this information can be used further to disambiguate terms. For example, given that the term "Course" is linked to terms like "Professor" and "University", it becomes clear that the "Course" here does not mean a "Golf course" or a "Course of a meal". Therefore, an ML model should be able to use all these various sources of semantics to extract and learn data-driven patterns that can be later used for modeling applications. However, most of the existing works do not sufficiently utilize these sources of semantics.

A. Conceptual Challenges

There are several conceptual limitations of the existing ML4MDE solutions —

NL Semantics not utilized (CL_1): most of the existing works in ML for MDE (ML4MDE) do not sufficiently utilize the NL "meaning" of lexical terms, ignore word order and the context in which they appear, treating words as isolated entities which leads to a loss of "meaning". Bag-of-words or static embeddings throw away word order and context, so polysemous labels like Course or Role are poorly encoded. Contextual encoders like BERT recover semantic cues, and ignoring them weakens the signal, i.e., the NL contextual semantics that the ML model can learn from the conceptual models [18], [19].

Modeling Language Semantics not utilized (CL_2): modeling language terms such as "EClass", "EPackage" in Ecore or "Node", "Software System" in ArchiMate or ontological keywords like "kind", "subkind" in OntoUML are ignored and only element "names" are used during training. Metamodel ontological primitives (e.g., EClass, EReference, kind, role) encode *constraints* and *types* that act as strong inductive bias for tasks like partial model completion or conformance checking. When ML model training ignores these semantics and uses only names, models miss the structure needed for valid predictions and generalization for that modeling language. Foundational ontologies and metamodel semantics are explicitly recommended to improve conceptual-model quality and reasoning [17].

Model's graph structure not integrated with NL semantics (CL_3): GNNs are used to capture topology, but rarely combined with contextualized text embeddings from the lexical terms of nodes and edges. Polignano et al. [20] show that combining graph embeddings along with contextualized word embeddings performs better than taking them individually.

Graph structure of models not leveraged to generalize across multiple modeling languages (CL_4): Existing works are typically focused on a single modeling language and cannot leverage the shared graph formalism across modeling languages [10]. Due to this, existing approaches do not provide a generic ML4MDE solution that is applicable or extensible to any new modeling language.

B. Technical Challenges

Furthermore, there have been certain technical limitations in the existing works —

Limited LLM finetuning (TL_1): Off-the-shelf pretrained language models, transformer-based models are rarely utilized and adapted to modeling vocabularies or domains. Leveraging the natural language semantics captured within a pretrained LLM and performing fine-tuning is directly linked to better downstream performance [21], [22]. However, existing literature has not leveraged the NL semantics captured by pretrained LLMs and adapted them for the conceptual model's text (metamodel, ontological terms).

OOV Problem (TL_2): Conceptual models can have a lot of new identifiers, camelCase tokens, and domain acronyms. Marcen et al. [7] point out that traditional word embedding approaches cannot represent unseen model-specific terms due to the out-of-vocabulary problem. Traditional word embeddings map unseen tokens poorly, degrading training signals. Subword methods (Byte Pair Encoding/fastText) mitigate but don't eliminate domain-specific OOV issues. Code/identifier literature shows domain-adapted tokenization/pretraining helps with invented labels, an analogue to model element names [23], [24].

Inflexible data configuration (TL_3): Practitioners cannot easily choose which model elements or attributes to include in the ML input. Elements can have attributes like "Student" having a "name" or "sr.no". Including irrelevant/noisy fields or excluding informative ones hurts the accuracy of ML models. Feature-selection research and ML systems guidance emphasize configurable inputs/ablations as critical for performance and maintainability [25]. Therefore, the modeler should be able to decide what textual information can be considered relevant during training an ML model for a given ML4MDE task.

Based on the aforementioned limitations of the existing works, in this paper, we aim to test the hypothesis H —

Learning contextually rich representation of conceptual models leads to performance improvements on ML4MDE tasks. To that end, we present **G**raph **L**anguage **M**odeling for **M**odel-**D**riven **E**ngineering (GLM4MDE), an end-to-end approach to train deep learning models to assist a modeler in tasks such as PMC and MCC. We aim to learn embedded modeling patterns directly from a corpus of models. Accordingly, we focus on tasks that can be trained solely on model-internal data, namely, Partial Model Completion (PMC) and Metamodel Classification and Clustering (MCC). We deliberately exclude tasks that require external inputs, such as automated domain model extraction from natural-language requirements, because that information lies outside the model dataset.

Concretely, the core contributions of this paper are: i) a generic approach to finetune LLMs that capture the linguistic aspects from NL, modeling language and ontological semantics; ii) Integration of GNNs that enrich the semantics using graph-structural information on a given dataset of models; iii) support for multiple modeling languages such as Ecore, OntoUML and ArchiMate while enabling extensibility for any new modeling language, iv) empirical and extensive evaluation of our framework on PMC and MCC using four datasets, namely Ecore555[2], ModelSet[3], OntoUML[4] dataset and EA ModelSet[5] [26] and comparative evaluation with traditional ML baselines; v) ablation studies for the different sources of semantics on the performance; and vi a python package which can support developers in ML4MDE that can be accessed via python package manager pip[6].

The remainder of the paper is structured as follows — Section II provides a brief overview of the necessary background as a precursor to our work. Section III provides an in-depth analysis of the works related to our approach that motivate our design decisions. Section IV provides the technical details of our framework. In Section V, we evaluate the performance of our framework. In Section VII, we discuss the implications of our results and the threats to validity of our approach. We conclude the paper with conclusion and future work in Section VIII.

II. Background

In the following, we provide a brief overview of the core concepts involved in our work. We describe what kind of models we deal with and which we use as data to train ML models. Next, we describe a core distinction between the partial model completion task as accomplished in the literature and how we achieve it. Then, we describe the notion of conceptual knowledge graphs (CKG), text classification using pretrained language models, and training graph neural networks on graph data.

[2]https://zenodo.org/records/2585456
[3]https://modelset.github.io/
[4]https://github.com/OntoUML/ontouml-models
[5]https://me-big-tuwien-ac-at.github.io/EAModelSet/home
[6]https://pypi.org/project/glam4cm/

A. Model

In the following, we formally define models that our framework considers valid inputs to train ML models. A graph-based model $\mathcal{G}$ is defined as a labeled, attributed graph:

$$\mathcal{G} = (V, E, \lambda_V, \lambda_E, \rho_V, \rho_E), \tag{1}$$

Where:

- V is a finite set of nodes (vertices),
- $E \subseteq V \times V$ is a finite set of directed or undirected edges,
- $\lambda_V : V \to \Sigma_V$ is a labeling function assigning labels from an alphabet Σ_V to nodes,
- $\lambda_E : E \to \Sigma_E$ is a labeling function assigning labels from an alphabet Σ_E to edges,
- $\rho_V : V \to \mathcal{P}(\mathcal{A}_V)$ is a function assigning a set of properties $\mathcal{A}_V$ to each node,
- $\rho_E : E \to \mathcal{P}(\mathcal{A}_E)$ is a function assigning a set of properties $\mathcal{A}_E$ to each edge.

The model has the following properties:

- Each node and edge has an associated label, usually represented by the 'name' property in a model.
- Each node and edge can have multiple attributes represented as key-value pairs. 'Type' is one of the keys that come from the modeling language.
- The structure of the model allows for flexible representation of various domains such as knowledge graphs, semantic networks, and social networks.

This formalism provides a structured representation of graphs where nodes and edges carry labels and properties, making it a generic formalism applicable to multiple modeling languages.

B. Partial Model Completion

Next, we provide a crucial distinction between the partial model completion task as it is accomplished in the literature and the way our approach defines it. While both approaches involve predicting missing information for a given partial model, PMC in the literature primarily involves predicting the *names*, i.e., natural language semantics, which requires ***generating*** terms consistent with the partial model's domain semantics, i.e., domain modeling [8]. Furthermore, in the literature, PMC is limited to predicting the names of the classes, attributes, and relationships. However, there has been no research involving link prediction (LP), i.e., predicting if there exists a link between two elements of a model (e.g., classes) or link classification (LC), i.e., predicting the type of the link between two elements (e.g., classes) in models. Therefore, overall, our framework aims to enable PMC such that a trained ML model can predict the missing modeling language and ontological semantics in a model in both nodes (e.g., classes in UML, elements in ArchiMate) and edges (relations) of the models. There are several reasons for this distinction.

Firstly, existing works primarily focus on domain modeling using Generative AI capabilities of Large Language Models (LLM) like ChatGPT[7], Llama[8], DeepSeek[9], and have been

[7]https://chat.openai.com/
[8]https://www.llama.com/
[9]https://chat.deepseek.com/

provided a designated research area, LLM4MDE [8], [27]. Given that the works in LLM4MDE only include generative models, we will denote the LLM4MDE research area as **GenAI4MDE**. GenAI4MDE approaches only aim at the conformance of their predictions with the domain semantics and do not focus on the modeling language or ontological semantics, which are highly crucial for a model. For instance, in Fig. 1, existing works would focus on predicting a class "Person" and not focus on the modeling property like "abstract" which indicates if the class is an abstract class or not or ontological property like "kind" or "subkind" or "role" that carry significantly different semantics. In principle, generative approaches can also be used to predict the modeling or ontological property; however, this would involve relying on the modeling data that the LLM is trained on, but this approach is neither reliable nor transparent. E.g., a modeler would need to *hope* that the LLM is trained sufficiently well to have *learned* the syntactic rules of the modeling language of her choice and applies those rules sufficiently well while predicting the modeling or ontological properties. However, existing research shows that current LLMs struggle with reasoning capabilities involving formal rules, even if the rules are simple [28]. Furthermore, all existing works in GenAI4MDE use well-known modeling languages that are sufficiently well available on the internet, e.g., UML in PlantUML specification [8], [29], [30], Goal Models [31], or ER Models [32]. Therefore, a reasonable approach that utilizes the language modeling capabilities of LLMs would be to fine-tune them on our model dataset.

Secondly, fine-tuning an LLM demands high computing costs and power. Modern LLMs possess billions to trillions of parameters, necessitating extensive time and resources for training. For instance, the estimated cost of training OpenAI's GPT-4 model exceeds $100 million, rendering it financially prohibitive for most small-to-medium-sized enterprises and the academic community [33]. However, it is important to note that our work does not require the LLM to learn to predict open-ended domain concepts, as in the case of domain modeling in existing works. Our approach requires the LLM to learn to predict modeling and ontological concepts with a much smaller and closed (fixed) vocabulary. For example, there are fewer than 30 ontological categories, only about 64 ArchiMate node types, and 11 relationship types. This enables us to do trade-offs and use much smaller LLMs like BERT (about 330M parameters) for finetuning.

Thirdly, while using LLMs, a critical issue of hallucination exists where the LLM produces seemingly plausible but incorrect responses. This is a hard problem to deal with because of the non-deterministic nature of this issue [34]. However, our approach avoids this issue because it does not use LLMs as generative models but as classification models from a fixed set of classes, limiting the output to that predetermined set. The output from our approach can be incorrect, but it cannot be unexpected.

While we differ quite significantly from existing GenAI4MDE works in accomplishing PMC like [8], [35], both approaches can complement each other with their strengths to create an advanced modeling assistant. The focus of this paper is not integration of the two approaches; however, we discuss how the integration could look like in Section VII.

C. Model Classification and Clustering

The second modeling task that we focus on is domain classification. This task aims to classify a model into a particular domain from a set of known domains. The Model Classification task is correlated to model clustering and provides almost identical results as model clustering, as seen in [36]. This task requires developing a similarity function f to determine the similarity between the elements of any two given models. Once f can determine the similarity between any two elements of the two models, this similarity can be aggregated to evaluate the similarity between models as a whole. Finally, based on the degree of similarity, models can be classified into the same or different domains or clustered together. Traditionally, works that did not involve embeddings developed several techniques to design a suitable f for similarity calculation. However, with embedding-based approaches that can represent a model as an aggregated vector of all the embeddings of its elements, cosine similarity or the inverse of Euclidean distance forms a straightforward choice of f. Once f is decided, in the case of model classification, a ML model is trained to classify the domain of the model. In model clustering, a clustering algorithm such as K-means [37] clusters the models into a given number of clusters.

D. Conceptual Knowledge Graph

Knowledge Graphs represent a collection of interlinked descriptions of entities – e.g., objects, events, and concepts. KGs provide a foundation for data integration, fusion, analytics, and sharing [38] based on linked data and semantic metadata. KGs have been recently used for the representation [39], [40] of models. Such KG-based representations can act as the conceptual representation of models to enable ML-based applications using models conforming to different modeling languages. Ali et al. [41] defined *Conceptual Knowledge Graphs* (CKG) are termed as "KGs representing Models".

E. Text Vectorization

Text vectorization converts NL into numerical form so that ML models can understand and process it. Since models work with numbers, not words, vectorization bridges the gap between raw text and ML algorithms. Text vectorization aims to enable text-based ML tasks like sentiment analysis, spam detection, recommendation systems, transform unstructured text into structured numerical representations, and capture meaning, structure, and relationships between words.

A vectorization technique takes texts as input and returns a fixed-size vector. There are several vectorization techniques with increasing implementation complexity and their ability to capture the "semantics" of the text. Fig. 2 compares various vectorization techniques in the literature. One-hot encoding is the most straightforward technique to implement, but leads to very sparse vectors of the size of the entire vocabulary. This

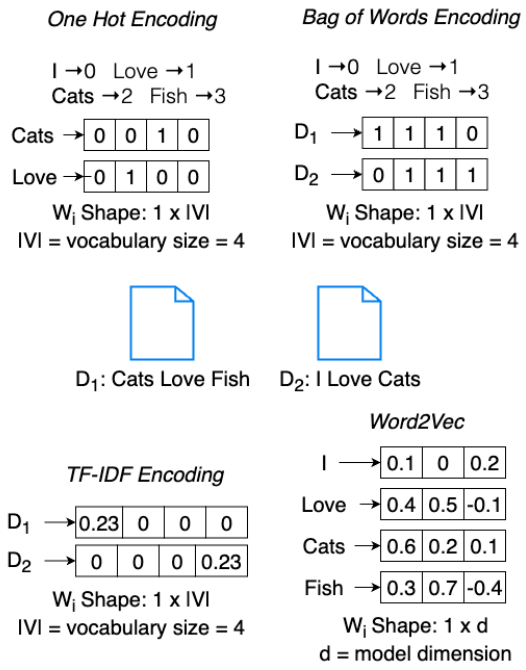


Fig. 2. Text Vectorization Techniques

approach does not consider word order or contextual words. Therefore fails to capture the "meaning" of the sentence. An advanced vectorization approach, word2vec, produces fixed-sized vectors that capture some semantic similarity between words.

F. LLM-based Text Classification

Text classification refers to the task of assigning predefined categories to textual data. Traditional approaches relied on rule-based methods and classical ML algorithms, such as Naïve Bayes, Support Vector Machines (SVM), and logistic regression [42], [43]. However, these methods often required extensive feature engineering and struggled with complex linguistic structures. Deep learning, particularly transformer-based pretrained language models (LLMs) [44], enabled contextualized word representations and captured long-range dependencies. LLMs leverage self-supervised learning on large text corpora, followed by fine-tuning specific classification tasks, leading to state-of-the-art performance in various NLP benchmarks. LLMs are deep neural networks that undergo two primary training phases —

Pretraining is conducted on a large-scale, unlabeled text corpus where the language model is trained over this corpus to learn generalized knowledge about natural language words. Bidirectional Encoder Representations from Transformers (BERT) is one of the LLMs that utilize the context of a word w in a sentence from both sides, i.e., words to the left and right of w to create a representation of w. In essence, the LLM learns to provide a contextually enriched representation of a text, i.e., embeddings, which can then be used with a classifier that can learn to classify the text into a category. An LLM can convert a text of fixed length, called an LLM's context length, into a vector. BERT has a context length of 512. Therefore, documents larger than 512 tokens[10] need to be split into documents of 512 tokens each and then aggregated to get the vector of the entire document. This can lead to some information loss. With recent advances in ML, ModernBERT [45], a modernized BERT-style LLM was released that supports a long context length of 8,192 tokens, i.e., 16x longer than BERT, making it ideal for tasks that require processing long documents.

Fine-tuning adapts a pretrained model to a classification task using labeled datasets. Common fine-tuning strategies include:

- **Single-Sentence Classification**: Mapping a single input text to a label (e.g., sentiment analysis).
- **Pairwise Classification**: Evaluating relationships between two texts (e.g., natural language inference).
- **Multi-Label Classification**: Assigning multiple labels to a single input (e.g., multi-topic categorization).

It is important to note that LLMs such as ChatGPT, Llama, and DeepSeek-R1 are decoder LLMs that generate text in an auto-regressive fashion, i.e., they predict each subsequent token based on previously generated tokens and the prompt. This makes them well-suited for generative tasks like open-ended text generation, summarization, or translation. However, the LLMs we use in our work are encoder-only (e.g., BERT) that focus primarily on transforming an input sequence into a contextual representation without generating new text token by token. These LLMs enable deep understanding of input text for tasks like text classification, extraction, and semantic matching—by encoding the entire sequence into meaningful embeddings. Studies [46], [47] have empirically shown that encoder-only models are more suitable for tasks involving natural language understanding (NLU) than decoder-only LLMs. Given that in our work, we use LLMs to capture the natural language semantics of the text, we use encoder-only LLMs.

G. GNN-based Representation Learning

Graph Neural Networks are neural models that learn graph representations via *message passing* between graph nodes. A node aggregates information from its neighborhood. GNNs leverage the inherent structure and connectivity of the graph to make informed predictions about the labels or properties of individual nodes. Thereby, GNNs capture the local and global context of each node, allowing them to assign appropriate labels based on the learned representations. This approach is particularly valuable in diverse applications, such as social network analysis, recommendation systems, biology, and knowledge graphs, where nodes represent entities and relationships. Recently, variants of GNNs such as Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), and Graph Recurrent Networks (GRN) have demonstrated good performance on many deep learning tasks. GraphSAGE [48] generalized the aggregation function (compared to GCN, which uses "mean" as the aggregation function) that generates node embeddings by sampling and aggregating features from a node's local neighborhood. GNNs can be combined with embeddings from LLM-based BERT embeddings

[10] 1 token $\approx$ 0.75 word

such that embeddings capture node-specific information, and the GNN can propagate the information by leveraging the graph structure. GNN's can be trained for classification tasks related to graphs such as i) node classification, which we use for node's modeling language and ontological semantics prediction, e.g., predicting the stereotype information in partially completed OntoUML models, ii) link prediction which we use to predict the presence of links between two nodes in a model, iii) edge classification, which we use to classify the type of the relationship between two nodes in a model, and finally, iv) graph classification, which we use for classifying the domain of a dataset of models.

III. Related Work

In the following, we provide a deep dive into how the existing works have used the information in the models to train ML models. The objective is to consolidate the learning from all these works and provide a rationale for the design choices of GLM4MDE. To outline the research gaps, we provide a comparative analysis of existing works for three MDE tasks along four dimension, namely, i) ***NLS***: whether natural language semantics of words are used indicated by the usage of any ontology like WordNet [49] or a pretrained language model, ii) ***MLS***: whether the modeling language semantics are used, iii) ***GS***: whether the graph structure of the models is used, and iv) ***FT***: whether the approach uses finetuned language models (FTLM) (only applicable if the approach uses i) with a pretrained language model). Next, we address the works in GenAI4MDE; however, we do not go in depth because these works focus on domain modeling, while we focus on aligning an existing partially completed model with modeling language and ontological semantics. Moreover, these works rely entirely on the capabilities of off-the-shelf LLMs while we finetune pretrained language models and GNNs on our datasets.

TABLE I
Semantic sources analysis of Model Clustering

Paper	NLS	LM	MLS	GS	FT	MSF
[50] (2013)	✗	✗	●	✓	NA	Distance based on Relationship Type
[51] (2016)	✓	✗	●	✗	NA	WordNet synset based similarity [52]
[53] (2016)	✗	✗	✓	✗	NA	Cosine Similarity with TFM, Dice coefficient [54], EMFCompare [55] Matching
[56] (2016)	✓	✗	✓	✗	NA	Cosine similarity based on relation type weight, tf-idf, WordNet similarity of terms
[57] (2017)	✗	✗	●	●	NA	Strict and relaxed n-gram matching
[16] (2024)	✓	✓	✗	✗	✓	Cosine similarity on word2vec embeddings

✓: Supported, ✗: Not Supported, ●: Partially
MLT: Machine Learning Training, TV: Text Vectorization
NLS: Natural Language Semantics, LM: Language Model
MLS: Modeling Language Semantics, GS: Graph Structure
FT: Finetuning, MSF: Models Similarity Function
NA: Not Applicable

TABLE II
Semantic sources analysis of Model Classification

Paper	MLT	NLS	LM	MLS	GS	FT	TV
[58] (2021)	✗	✗	✗	✓	✗	NA	tf-idf
[59] (2021)	✓	✗	✗	●	✗	NA	n-grams tf-idf
[60] (2022)	✓	✗	✗	✗	✓	NA	Graph Kernels
[36] (2024)	✓	✓	✓	✗	✗	✓	Word2vec, BERT tokenizer

✓: Supported, ✗: Not Supported, ●: Partially
MLT: Machine Learning Training, TV: Text Vectorization
NLS: Natural Language Semantics, LM: Language Model
MLS: Modeling Language Semantics, GS: Graph Structure
FT: Finetuning, MSF: Models Similarity Function
NA: Not Applicable

A. Model Clustering

In the following, we discuss the works related to model clustering. In Table I, we show comparative analysis along the aforementioned four dimensions and briefly describe the kind of similarity function used by these works to determine whether two models can be clustered together. In [50], Struber et al. construct a distance function based on the inverse of the strength of the link, where a containment relation has a stronger link than a simple relation link. The higher the distance, the lower the similarity. This similarity function is then used to cluster similar nodes together. In their work, the authors partially used the modeling language semantics by treating the type of links differently and also considered the graph structure of the model. In [51], Elkamel et al. define the similarity between two models based on the similarity between the names, attributes, and operations across all pairs of classes in two models. In their work, the authors leveraged WordNet ontology to use the terms' natural language semantics. Elkamel et al. [51] did not consider relationship type separately and only focused on comparing classes. In [53], Basciani et al. use EMFCompare [55] to find the similarity of two models. EMFCompare considers the modeling language type information; therefore, this work uses the modeling language semantics in its similarity function. In [57], Babur et al. design n-grams, i.e., they represent each class in a model not as a single word. In [16], a word2vec model trained on data specific to MDE is used to generate word embeddings, and then cosine similarity is used to compare the models.

Based on the Table.I, we note the transition of model comparison approaches from using simple graph-based or ontology-based (WordNet) approaches to advanced word embeddings approaches to capture natural language semantics [16].

B. Model Classification

In Table II, we look at the model classification works and analyze the sources of semantics contained within a model that are used as input to a classifier that categorizes a model into one of the predefined categories. This task generally involves a dataset D and an input query model q. This task aims to

create a classifier f that can categorize q accurately into one of the predefined categories C. In [58] Rubei et al. create a tf-idf index using only the names from the UML model's classes, references, and class attributes. The created index is used to classify q. First, the index transforms the model into a tf-idf vector, then it finds the top n most similar models, i.e., *hits*. Finally, the category of the most similar model among the n retrieved models is assigned as the category of q. In this process, there is no learning or ML model training involved. Nguyen et al. [59] build on top of the approach in [58] to create a tf-idf index. However, given that Rubei et al. in [58] used only names of elements, which led to ignoring the context of an element, [59] involved n-grams of each model element to consider the context. E.g., for an attribute "title" of a class "Book", n-gram representation of "title" is "Book.title.EString.0.1", which has 5-grams and includes the class name, data type, and cardinality of the relationship between class "Book" and the attribute "title". This n-gram approach provides more context about the model element. A TF-IDF index is created using the n-grams to vectorize the models. Given that D is a labeled dataset, the model vectors are then used to train a neural network classifier that learns to classify the model into the correct category accurately. In Khalilipour et al. [60], the authors leverage only the structural features of the models transformed into a set of graphs, to use as features that ML models further use to classify the graph. This approach uses Graph Kernels, which are mathematical functions that measure the similarity between two graphs. E.g., Random Walk kernel counts matching graph traversals or local neighborhood similarities in two graphs [61] or Weisfeiler-Lehman [62] that uses node label information of neighboring nodes to represent a node. Graph Kernels vectorize a model, and these generated vectors can be used to train an ML model for model classification. [60] train Support Vector Machines (SVM) [63], Random Forest (RF) [64] and Artificial Neural Networks as classifiers. In [16], [36], Lopez et al. use pretrained word2vec and BERT language models to transform a model into a vector and then classify it. These approaches use NLS captured by the pre-trained language models trained on a large natural language corpus.

Based on the Table.II, we note that the graph structure is hardly used in model classification tasks. In [65], Lopez et al. conclude that the *names* of elements within a model are more suitable for model classification than the model's graph structure. While this conclusion seems reasonable for using *names* of elements within a model and graph structural information separately, their work neither considers nor comments on the effect of using both NLS and GS combined for model classification. We further note in Table II that the simple text vectorization techniques from tf-idf and n-grams are replaced gradually with language models like word2vec or BERT.

C. Partial Model Completion

In the following, we discuss the works relevant to partial model completion. While this task is covered more extensively using LLMs in Section III-E, we evaluate the semantic sources used to predict a new element in a partially completed model in the following.

TABLE III
SEMANTIC SOURCES ANALYSIS OF PARTIAL MODEL COMPLETION

Paper	MLT	NLS	LM	MLS	GS	FT	TV
[66] (2021)	✗	✓	✓	●	✗	✗	Word2Vec
[9] (2022)	✓	✓	✓	✓	●	✓	BERT Tokenizer
[67] (2023)	✗	✗	✗	●	✗	NA	Package, Class, Attributes Relationship Matrix
[68] (2023)	✗	✗	✗	✗	✓	NA	Term Frequency Matrix

✓: Supported, ✗: Not Supported, ●: Partially
MLT: Machine Learning Training, TV: Text Vectorization
NLS: Natural Language Semantics, LM: Language Model
MLS: Modeling Language Semantics, GS: Graph Structure
FT: Finetuning, MSF: Models Similarity Function
NA: Not Applicable

Burgueno et al. [66] present an approach that uses pretrained language models like Word2Vec and general NLP techniques like morphological analysis and lemmatization of words to create a recommender system that recommends the following elements, like classes, relationships, or attribute names for a given partially completed model. Burgueno et al. do not train any ML model in this work and use the NLS captured by the pretrained word2vec language model trained on Twitter, Wikipedia, or Google News data. In [9], Weyssow et al. fine-tune a BERT model on a model task to predict the names of a model's classes, attributes, and relationships.

D. Qualitative Assessment.

Beyond the comparative overview in Tables I–III, a qualitative analysis highlights both the strengths and shortcomings of existing ML4MDE works. Early efforts such as [50], [51] showed that even limited use of semantics (e.g., differentiating link types or exploiting WordNet-based similarity) could improve clustering outcomes, while Babur et al. [56], [57] advanced this further by incorporating richer textual units such as n-grams. More recent studies [16], [36] successfully demonstrated that contextualized embeddings capture subtleties overlooked by bag-of-words or tf-idf, and Weyssow et al. [9] provided evidence that fine-tuning BERT on modeling data yields stronger recommendations in partial model completion.

However, most approaches suffer from systematic omissions that can potentially limit their effectiveness. Natural language semantics are often underutilized ($\mathbf{CL}_1$), with many clustering and classification methods still relying on static embeddings or tf-idf [53], [58], [59], resulting in poor handling of polysemous labels (e.g., "Course" as education vs. meal). Modeling language primitives such as `EClass` or `kind` are neglected ($\mathbf{CL}_2$), even though they encode inductive biases crucial for valid predictions; for instance, PMC approaches like [66] rely on general-purpose embeddings without leveraging these constraints. While graph kernels and GNNs capture topology, they are rarely combined with textual semantics ($\mathbf{CL}_3$), leading to purely structural or purely lexical solutions without multimodal enrichment [60], [68]. The graph structure is rarely used as a unifying abstraction across multiple modeling languages ($\mathbf{CL}_4$), limiting generalizability. On the technical

side, most works employ off-the-shelf embeddings without domain adaptation ($\mathbf{TL}_1$), thereby missing the gains from fine-tuning LLMs on modeling vocabularies; they also struggle with out-of-vocabulary identifiers such as camelCase tokens or acronyms ($\mathbf{TL}_2$), and rarely allow configurable inclusion of model attributes during training ($\mathbf{TL}_3$). Consequently, existing ML4MDE approaches demonstrate partial semantic exploitation but fail to fully integrate natural language, modeling language, and structural semantics. These gaps motivate our framework, GLM4MDE, which explicitly addresses $\mathbf{CL}_1$–$\mathbf{CL}_4$ and $\mathbf{TL}_1$–$\mathbf{TL}_3$ by jointly leveraging fine-tuned LLM embeddings and GNN-based structural learning to produce semantically rich and structurally grounded model representations.

E. GenAI4MDE

The emergence of LLMs has led to the application of LLMs in MDE, specifically domain modeling, i.e., generating or completing partial models from natural language descriptions. Several works [8], [30], [35], [69] have shown promising results in generating models from textual descriptions of requirements. However, these approaches involve a few limitations. Firstly, these works rely on the modeling capabilities of the LLM and knowledge about modeling languages that the LLM came across in its training data. This limits the applicability of these approaches only to specific, well-known modeling languages. All the existing works use well-known modeling languages such as UML (particularly, class diagrams), ER models. Existing works assume that the LLM "knows" the formal rules of modeling language; however, with a less-known modeling language, an LLM may fail to maintain strict logical consistency and miss important constraints during domain modeling. Secondly, fine-tuning (adapting) an LLM for a new domain dataset is computationally and financially expensive. Thirdly, these works assume a simple metamodel with only classes, relationships, and attributes, and rarely consider the type information of these constructs. These works do not distinguish between "containment" and "reference" relationships or "abstract" and "concrete" classes. Therefore, the focus is primarily to make sure the generated models are meaningful domain models and do not focus on the model's conformance with rich and complex modeling language semantics.

F. Synopsis

Based on the existing related works, we note that none combine all three sources of semantics, i.e., NLS, MLS, and GS. Finetuning the language models is also very scarce. None of the works generalizes their approach to multiple modeling languages and only focuses on UML class diagrams. Furthermore, none of the works provide support for configuring what sources of semantics are utilized to train an ML model. We note that Lopez et al. [16] also present a framework for training ML models for MDE. While the overall objective of the framework is similar to ours, we outline the key differences between [16] and our work in Table IV. The table presents different criteria along the same lines of conceptual and technical challenges that differentiate our work from [16] and consequently underpin our framework's need and motivation.

TABLE IV
COMPARISON WITH MODELXGLUE EXISTING ML4MDE FRAMEWORK

Criteria	ModelXGlue [16]	Ours
Tasks		
Model Classification	✓	✓
Model Clustering	✓	✓
Modeling Element Recommendation	✓	✗
Modeling Element Type Classification	✗	✓
Link Classification	✗	✓
Link Prediction	✗	✓
Semantic Sources		
Model Element Textual Labels	✓	✓
Modeling Language Types & Attributes	✗	✓
Graph Structure	✗	✓
LLM + GNN Combination		
GNN integrated in pipeline	✗	✓
Configurable Model→Text Serialization		
User-configurable textual representation	✗	✓
Multiple Modeling Languages		
UML class diagrams	✓	✓
ArchiMate	✗	✓
OntoUML	✗	✓
LLM Fine-tuning Support		
Finetuning on model data (HF LLMs)	✗	✓

✓ Supported ✗ Not supported ● Partially supported

IV. GLM4MDE Framework

In the following, we introduce our GLM4MDE framework. We first discuss the requirements of our framework, then we provide a design comprising all the steps involved in the end-to-end training of LLM and GNN on model datasets. Then, we provide details of each step.

A. Requirements

We provide the requirements that inform the design decisions and features supported by GLM4MDE.

R_1: The framework should support the integration of any LLM from the Huggingface LLM repository[11] to utilize the natural language semantics embedded in the lexical terms within a model. Furthermore, the framework should also support training simpler traditional language models, such as word2vec or encodings such as TF-IDF, for comparative analysis of these ML models, and let the ML expert select the most suitable one. This requirement aims to tackle CL_1 and TL_2.

R_2: The framework should utilize the modeling language semantics, i.e., the vocabulary of the modeling language that the model contains, and should not simply rely on the *names* of the model elements. This requirement is aimed at including modeling language-specific contextual semantics. This requirement tries to resolve CL_2.

R_3: The framework should utilize the models' graph structure combined with the NL semantics to create a contextually rich vector representation of a model instead of using either separately. This resolves CL_3.

R_4: The framework should be extensible to any modeling language that can represent its models in a graph-based formalism as defined in Eq. 1. This requirement fulfills CL_4.

R_5: The framework should support finetuning large language models on the natural language labels in the model's dataset. This requirement fulfills TL_1.

R_6: The framework should support configuring the data in a model used to train an ML model. This feature aims to provide control over "what" data to focus on during training. Furthermore, this feature can also enable explainability of the ML model by allowing users to see the effect of the data attributes used on the ML model's performance. This feature can enable understanding the importance of different data attributes of a model for a given modeling task, such as model classification or relationship prediction. This requirement aims to resolve TL_3.

B. High Level Design

In the following, we elaborate on the high-level design (HLD) of our GLM4MDE framework. We show the HLD in Fig. 3. The framework enables developers to train ML models on their own model dataset for prediction tasks, namely, i) Graph Classification (GC) enabling MCC, ii) Node Classification (NC) enabling PMC to predict the modeling language attributes of the model nodes, iii) Edge Classification (EC) enabling PMC to predict the modeling language attributes of the model edges, and iv) Link Prediction (LP) enabling PMC to predict the missing links between model nodes. The entire end-to-end workflow consists of five steps as shown in Fig. 3. Step one transforms a model's dataset into a set of conceptual knowledge graphs for each model. In the second step, relevant data is extracted from each model's nodes and edges, which will be used to fine-tune the LLMs for the model's dataset. In the third step, a pretrained LLM is finetuned to use the extracted data from the models for prediction tasks. In step four, we turn the text from step two into vector embeddings. Usually, we use the model fine-tuned in step three. However, one can skip fine-tuning and use a pretrained model (e.g., OpenAI's embeddings API) or a simple vectorizer like TF-IDF or word2vec to create the embeddings. The last step involves using the generated embeddings and the knowledge graph structure to train GNNs for the prediction tasks. We now elaborate on the details of each step.

C. Model to Conceptual Knowledge Graph

In this step, we transform models from any modeling language into a graph with a predefined data model. Fig. 4 shows the metamodel for transforming models in a given modeling language to a CKG. CKG is implemented as a NetworkX[12] [70] graph. NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. With NetworkX, we can create, load, and store graph structures. NetworkX provides simple and flexible functions for working with graphs. In Fig. 4, LangGraph is a directed NetworkX graph specializing in a NetworkX Graph of each modeling language. The metamodel enables extending support for new modeling languages by implementing the abstract method "construct_graph" that takes the JSON representation of a model and outputs the transformed NetworkX Graph of the model. Our framework supports transforming models from OntoUML, Ecore-based Class Diagrams, and ArchiMate as the modeling languages, as shown in Fig. 4. We utilize the data cleansing and deduplication pipeline for models as proposed by Djelic et al. [71]. The class "CKGDataset" denotes the dataset of transformed graphs. The dataset class consists of properties such as "min_edges", which control the minimum number of edges, and "min_enr", i.e., minimum edges to nodes ratio that a graph must have in order to be part of the dataset. These properties are included to filter graphs with too few edges that are not meaningful. We noted that the dataset contained dummy models with hundreds of isolated nodes. These properties filter out such graphs. Furthermore, a boolean property "remove_duplicates", if enabled, can remove duplicate models from the dataset.

LangGraph stores several key attributes that are useful during training the ML model. Each node and edge has a numeric ID in a LangGraph. These numeric IDs are helpful in efficiently accessing nodes and edges during ML model training. Fig. 5 shows an example of an Ecore model transformed into a CKG. A LangGraph consists of multiple elements, i.e., nodes and edges. Each graph element consists of a label, which stores the natural language label, such as "Professor",

[11] https://huggingface.co/models

[12] https://networkx.org/documentation/stable/tutorial.html

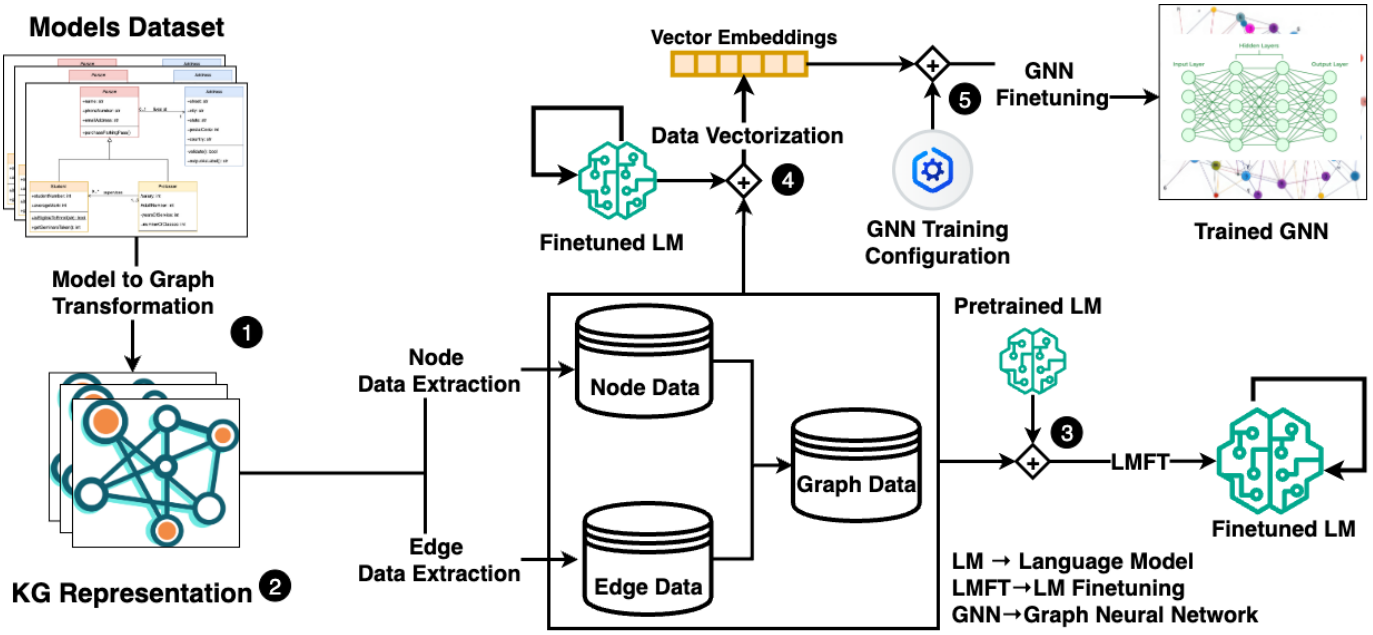


Fig. 3. High Level Design of GLM4MDE

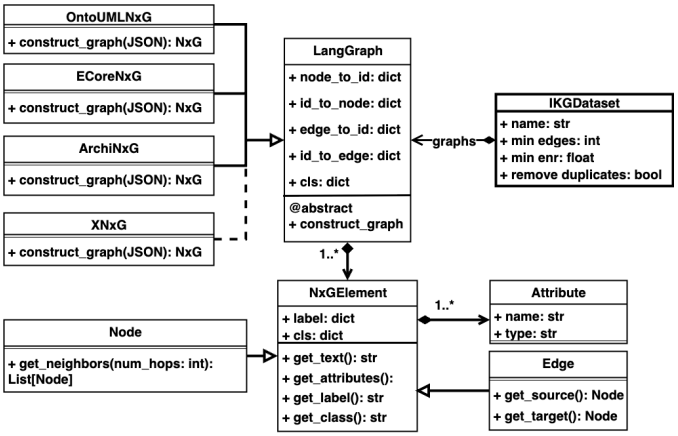


Fig. 4. Modeling Language Graph Transformation Metamodel

"Student" in Fig. 5. The method "get_label" allows access to the label of the graph element. Each element can have attributes. E.g., the "Person" class can have attributes such as "name", and "phoneNumber". A relationship "supervises" has an attribute "type" with value "reference". Each attribute has a name and a type. The method "get_attribute" provides access to the list of attributes of the graph element. The method "get_text" provides the textual representation of the element. This method is discussed in greater detail in Section IV-D. A node element can access the node neighbors up to k hops, i.e., nodes that can be reached by traversing k edges from the current node. The edge element can access its source and target nodes. LangGraph and NxGElement have a key attribute "cls" that indicates the attribute that the ML model is trained to predict based on the non-cls attributes. This "cls" attribute is discussed in more detail in Section IV-E.

D. Graph Data Extraction

After transforming the model dataset into a set of graphs, the next step involves extracting the relevant textual information from each graph. The extracted set of texts is used as the dataset to fine-tune the LLMs for the model's dataset. Fig. 6 shows the node data extraction based on several configurations that provide control over what information within a model will become part of the training data. This step allows an ML engineer to fine-grain control the LLM on specific aspects of the model dataset per the task requirements. E.g., in the case of a model domain classification task, using only the node and edge labels might be suitable, as Lopez et al. [65] suggest in their finding that the domain of the model mainly depends on the natural language terms present in the model. However, for tasks such as Partial Model Completion, Link Prediction, and Link Classification, relying only on node and edge labels is insufficient. These tasks require not only recognizing the lexical meaning of terms but also reasoning over structural dependencies (e.g., containment hierarchies) and modeling language semantics (e.g., metamodel constraints, ontological types). Without these additional semantic signals, the ML model may recommend elements that are lexically plausible but structurally invalid, mispredict relations that violate metamodel rules, or fail to disambiguate polysemous terms in context.

We transform the information captured by a graph node in a textual representation. The textual representation T_n of a node is built not only from the node's own information but also from the information of its neighbors up to k steps away. We provide the data extraction for the node "Person" in Fig. 6. Formally, we define the textual representation as:

$$T_n = \sigma(n) + \bigcup_{d=1}^{k} \bigcup_{m \in N_d(n)} \sigma(m) \tag{2}$$

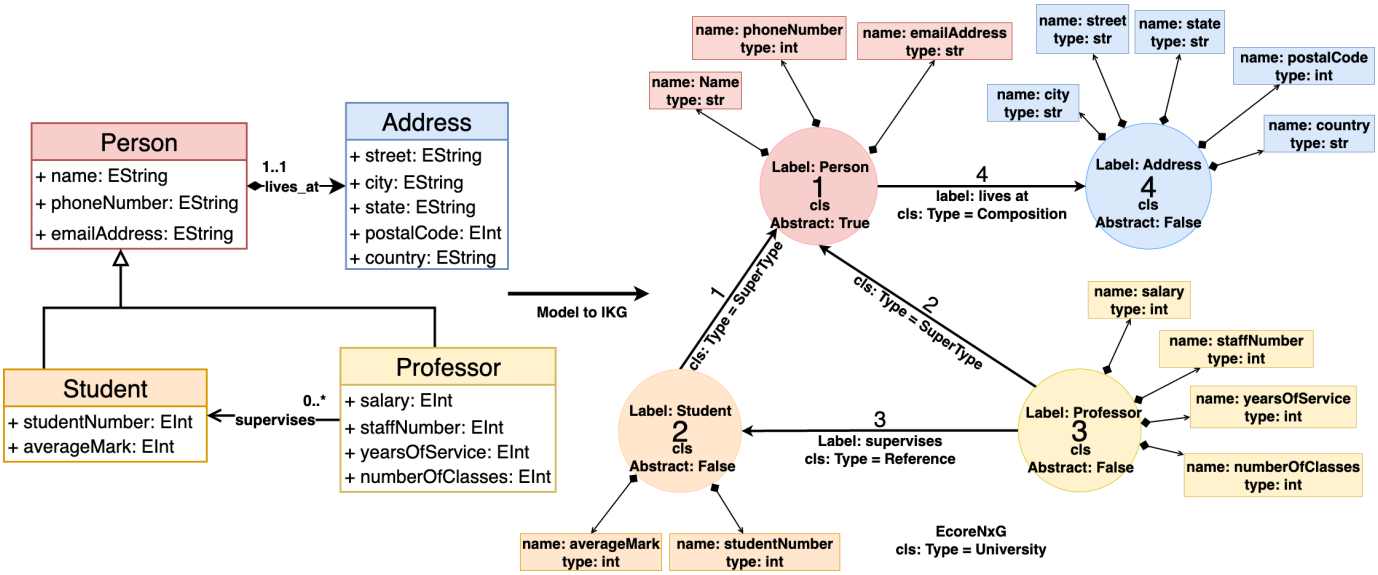


Fig. 5. Example Conceptual Model to Knowledge Graph Creation

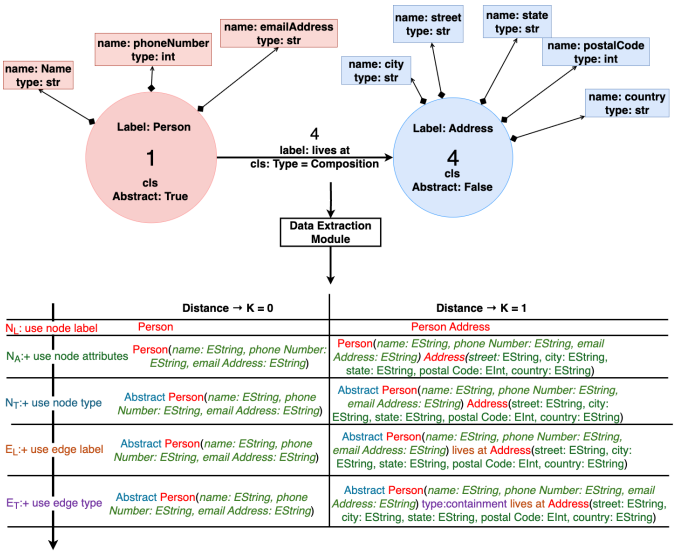


Fig. 6. Node Data Extraction

where σ is a function that determines what textual data to use from a node (e.g., its label, attributes, type, or edge information), and $N_d(n)$ denotes the set of neighbors of n at distance d. In other words, when $k = 0$, T_n contains only the data from n itself, whereas for $k > 0$, T_n also incorporates the data from neighbors within k hops.

For example, in Fig. 6, if we take the node "Person" with $k = 0$, its representation is limited to its own attributes, such as `Person(name, phoneNumber, emailAddress)`. When $k = 1$, the neighboring "Address" node and its attributes are also included, yielding `Person(name, phoneNumber, emailAddress) livesAt Address(street, city, state, postalCode, country)`. Thus, Eq. 2 captures how increasing k enriches a node's representation with contextual information, allowing the ML model to learn not only what a "Person" is, but also how it connects to other elements like an "Address." Including neighbor information in this way makes the representation more expressive, allowing the ML model to learn not just what a "Person" is, but also how a person is connected to other elements like an "Address."

The six control parameters gradually increase the contextual information used in the textual representation of a node. The default case (N_L) uses the node label information. N_A includes the information from the attributes. Then N_T includes node type information. In this case, the "Person" class is an "abstract" class therefore a keyword "abstract" is added to its node description. However, the "Address" class is a concrete class therefore no node type information is added to the node description of "Address". The edge label and edge type information usage becomes applicable when $k > 0$, i.e., when the neighbors are considered in T_n. In case of $k = 1$, the edge label "lives at" is added to T_n when E_L is enabled and "type:composition" is added to T_n when E_T is enabled.

$$T_e = \sigma(e_{src}) + \sigma(e_{dest}) \tag{3}$$

where e is an edge between node e_{src} and e_{dest}. Similar to node data extraction, we can extract the edge data for each edge. In order to create textual data for an entire graph, we can use T_n of all the nodes using a node data extractor or T_e of all the edges using an edge data extractor.

This control also enables an ML engineer to evaluate the importance and effect of different sources of information present in the model on the ML model's performance for a specific task. Given that explainability is a critical topic in ML-driven solutions and currently LLMs are treated as black boxes [72], this feature can help an ML engineer explain or understand certain crucial correlations between the performance of ML models and the data involved in training, thereby improving explainability.

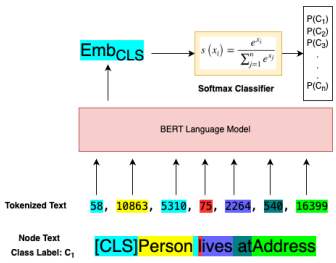


Fig. 7. Finetuning an LLM of Models Dataset

E. LLM Finetuning

Once the data is extracted from the models, one can fine-tune a suitable LLM on this data to familiarize the LLM with the dataset specific to this modeling language. Fig. 7 shows the LLM finetuning process using a text classification task. The process starts with raw text appended by a special "[CLS]" token. Then, a tokenizer transforms the words into a set of token IDs. Note that these are token IDs and not word IDs. Due to this, each token does not necessarily need to be a meaningful word. E.g., "ives" has a token ID 2264 even though "ives" is not a meaningful word. This happens because tokenizers are trained by analyzing vast amounts of text and breaking words into smaller pieces, i.e., tokens based on how frequently they appear in the entire corpus, such that each word can be transformed into a list of token IDs. Given "ives" is a common suffix in the English Language, it gets a separate token ID. This also avoids the OOV problem. Then, the tokens are passed through the LLM, in this case, a BERT encoder model. BERT outputs an embedding of each token, but we pick the embedding of the special "[CLS]" and use it as a proxy to classify the original text into its class C_1 using a softmax classification layer. After finetuning an LLM over this classification task, it gets accustomed to the modeling language dataset by adjusting its weights. This fine-tuned model can now generate text embeddings for text from the model's dataset of the same modeling language. There are other techniques to fine-tune an LLM. However, we chose this approach because it provides the benefit of LLM-based classification that can already be used for partial model completion. Ali et. al [10] show that in some cases, LLM-based finetuning is sufficient and does not need further GNN-based classification.

GLM4MDE uses LLM-based classification for all four of the ML4MDE. The value of the "cls" label in graphs is used as the label on which the LLM is trained. In Fig. 5, e.g., graph classification takes place on the "type" attribute of the graphs of the Ecore models dataset, which denotes the domain of the graph. The graph in Fig. 5 belongs to the "University" domain. In case of nodes, the "cls" field is "abstract" which denotes if the class is an "Abstract" class or not. In case of edges, the "cls" field is "type" which denotes the edge type out of "composition", "reference", or "super type". In the case of GC, we can use the extracted graph data from the data extraction step for classification. However, there are several tweaks needed to adapt the finetuning process for NC, EC, and LP tasks. In case of NC, the graph dataset is transformed into a dataset of texts of nodes using the data extractor and their corresponding "cls" labels. A fraction of these nodes are used for training the LLM and the rest are used as test nodes. Note that in case of NC, using the "use_node_types" configuration parameter (see Fig. 6) can contaminate the dataset, i.e., the label that the LLM is meant to predict is already part of the node texts. Therefore, to avoid that, if we have a node data T_n of node n, we ensure that even if "use_node_types" is turned on, we do not use the node type information in T_n. Further note that if $k > 0$, a training node can have a neighboring node that is a test node due to having training and testing nodes. However, we cannot have the node type information of test nodes be part of T_n of any node. Therefore, the node type of information of the neighboring nodes is removed if the neighboring node is a test node. The node type information of a training node is used when the training node forms part of T_n of a node. This information from the training node can help the LLM predict the node type using the node type of the nodes for which the node type is already known, i.e., training nodes. In case of EC, we create a dataset of T_e of all the edges in the entire dataset along with their "cls" labels. Again, if $k > 0$ and "use_edge_types" is set, we remove the edge type information for the test edges that form part of T_e of any edge. We further note that if $k > 0$ and the "use_edge_types" is set for Ecore models dataset for NC, then the edge type "supertype" is always connected to an abstract node. Therefore, to avoid such contamination of the dataset, we turn off the "use_edge_types" flag. Similarly, in the case of EC, we disable the use of node type because the presence of abstract classes is highly correlated with edge type "supertype". In such cases, the ML model will not learn patterns from the natural language text and instead learn these *easy hacks* to maximize classification results for specific types while disregarding some domain-specific crucial semantics provided by less frequent types.

F. Data Vectorization and GNN Finetuning

Once we have fine-tuned the LLMs on the data extracted from the graphs, we can get contextually rich embeddings of the texts. Fig. 8 shows the data embedding step. It is possible to use a pretrained, i.e., non-fine-tuned LLM, directly to get the node embeddings. However, such embeddings will lack the model's dataset-specific semantics as the pretrained LLM is trained on a generalized, domain-independent corpus on the internet. The LLMs provide a vector representation of texts. Note that in this step, any vector encoder, such as simpler language models like word2vec, TF-IDF vectorizer, can be used to transform a set of texts into vectors. However, such texts will lack contextual semantics. Using the node and edge embeddings, we can further fine-tune the node embeddings for NC and the edge embeddings for EC and LP using GNN-based fine-tuning.

Once we have the node embedding X, edge embeddings E, and the adjacency matrix A for all the sets of graphs, we can train GNN to utilize the graph structural information apart from the semantics information captured by the embeddings from the finetuned language models. We use this approach for GC, NC, EC, and LP tasks.

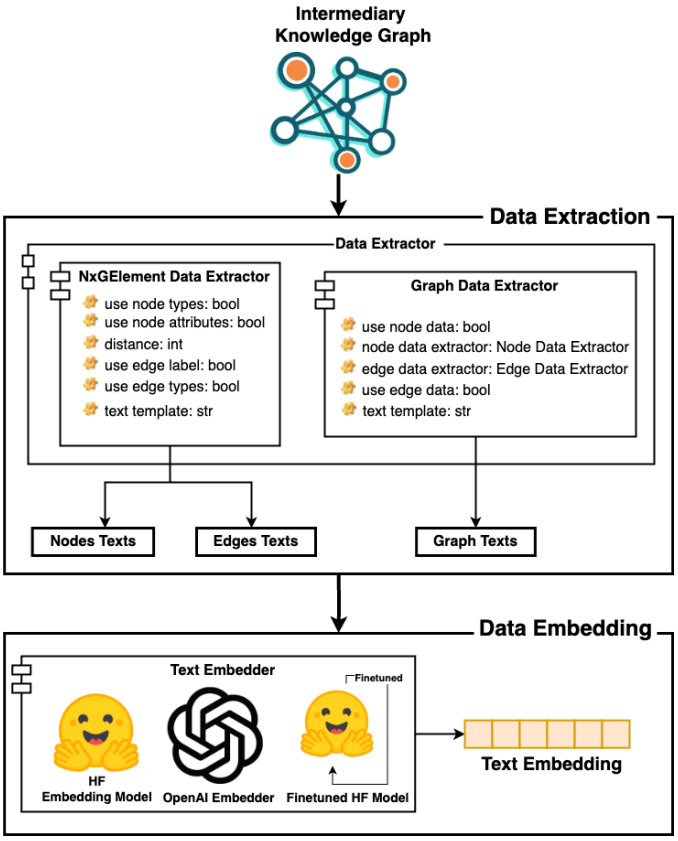


Fig. 8. Data Extraction Step for Graph Language Modeling

Given node embeddings X_i, edge embeddings E_i, and adjacency matrices A_i for n graphs, we define the GNN training tasks as follows.

1) Node Classification: The goal of node classification is to predict the label of individual nodes in a graph. The model learns node embeddings through iterative message passing, and a classification function is applied to predict the label of each node.

The node embeddings are updated as:

$$H^{(l+1)} = \sigma\left(A_i H^{(l)} W^{(l)}\right) \tag{4}$$

Where:

- $H^{(0)} = X_i$ (initial node embeddings),
- $W^{(l)}$ are trainable weight matrices,
- σ is a non-linear activation function.

The classification probability for a node v in graph i is:

$$\hat{y}_v = \text{softmax}\left(H_v^{(L)} W_{\text{out}}\right) \tag{5}$$

The loss function (cross-entropy loss) is:

$$\mathcal{L}_{\text{node}} = -\sum_{v \in \mathcal{V}_L} y_v \log \hat{y}_v \tag{6}$$

where $\mathcal{V}_L$ is the set of labeled nodes. The loss is propagated backwards into the neural network to update the neurons' weights in all layers and adapt the neural network to minimize the loss.

2) Edge Classification: Edge classification aims to predict the type or existence of edges between nodes in a graph. The model learns node representations and uses them to infer edge properties.

To classify edges, we compute the node embeddings as:

$$H^{(l+1)} = \sigma\left(A_i H^{(l)} W^{(l)}\right) \tag{7}$$

We define the edge embedding as:

$$H_{(u,v)} = f(H_u^{(L)}, H_v^{(L)}, E_{(u,v)}) \tag{8}$$

Where f is a function (e.g., concatenation or element-wise multiplication).

The edge classification probability is:

$$\hat{y}_{(u,v)} = \text{softmax}\left(H_{(u,v)} W_{\text{out}}\right) \tag{9}$$

The loss function is:

$$\mathcal{L}_{\text{edge}} = -\sum_{(u,v) \in \mathcal{E}_L} y_{(u,v)} \log \hat{y}_{(u,v)} \tag{10}$$

where $\mathcal{E}_L$ is the set of labeled edges.

3) Link Prediction: Link Prediction focuses on determining whether an edge should exist between two nodes. The model learns node embeddings and predicts missing or future links by computing the similarity between node pairs.

The node embeddings are learned as:

$$H^{(l+1)} = \sigma\left(A_i H^{(l)} W^{(l)}\right) \tag{11}$$

The probability of a link between nodes u and v is computed using a similarity function:

$$\hat{y}_{(u,v)} = \sigma\left(f(H_u^{(L)}, H_v^{(L)})\right) \tag{12}$$

where f can be:

- Inner product: $f(H_u, H_v) = H_u^\top H_v$,
- Concatenation: $f(H_u, H_v) = W \cdot [H_u \| H_v]$,
- Element-wise multiplication: $f(H_u, H_v) = H_u \odot H_v$.

The loss function is typically a binary cross-entropy loss:

$$\mathcal{L}_{\text{link}} = -\sum_{(u,v) \in \mathcal{E}_L} \left[y_{(u,v)} \log \hat{y}_{(u,v)} + (1 - y_{(u,v)}) \log(1 - \hat{y}_{(u,v)})\right] \tag{13}$$

Where $y_{(u,v)}$ is one if an edge exists and zero otherwise.

4) Graph Classification: Graph classification involves predicting a single label for an entire graph rather than individual nodes or edges. The model aggregates node representations to form a global graph embedding, which is then used for classification.

For whole-graph classification, node embeddings are computed as:

$$H^{(l+1)} = \sigma\left(A_i H^{(l)} W^{(l)}\right) \tag{14}$$

The graph-level representation is obtained via a readout function:

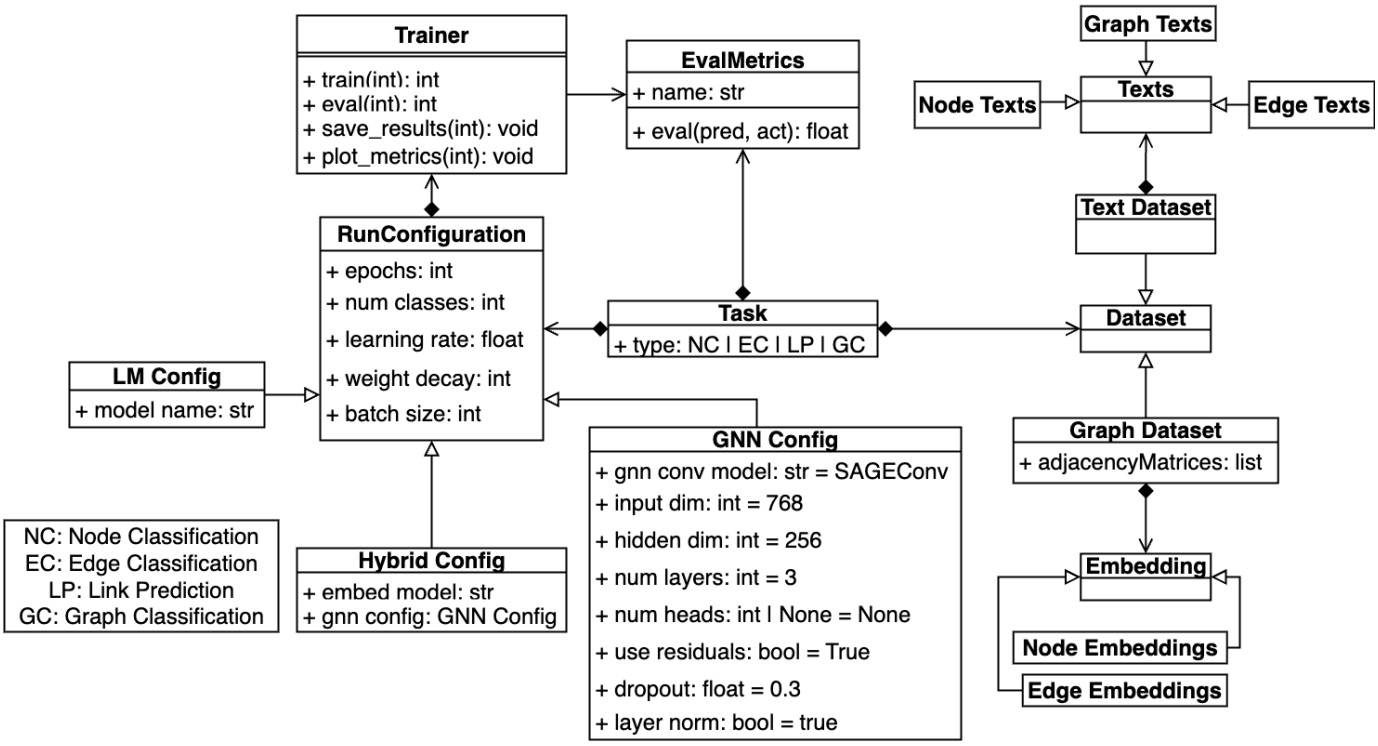


Fig. 9. Conceptual Modeling Tasks Support from GLM4MDE

$$H_G = \text{READOUT}(H^{(L)}) \tag{15}$$

Where READOUT is a permutation-invariant function (e.g., sum, mean, or max).

The graph classification probability is:

$$\hat{y}_G = \text{softmax}\left(H_G W_{\text{out}}\right) \tag{16}$$

The loss function is:

$$\mathcal{L}_{\text{graph}} = -\sum_{i=1}^{n} y_G \log \hat{y}_G \tag{17}$$

Where y_G is the true label for each graph.

Fig. 9 presents a model for training **Large Language Models (LLMs)** and **Graph Neural Networks (GNNs)** separately, as well as in a **hybrid** format. The system is designed to process datasets from **model-driven engineering (MDE) frameworks** such as *Ecore, ArchiMate, and OntoUML*, using structured and textual data extracted from these models. This architecture provides a **flexible and modular** approach to training LLMs, GNNs, and hybrid models on **model-driven engineering datasets**.

V. Evaluation Setup

We empirically evaluate our proposed framework on multiple classification tasks and datasets in the following. Furthermore, we evaluate the effect of different configurations regarding the data involved in training the ML models by providing ablation studies on all tasks.

A. Research Questions

In order to evaluate our framework, we focus on the following research questions —

- **[RQ1]** What is the performance of GLM4MDE on the modeling tasks node classification, edge classification, link prediction, and graph classification?
- **[RQ2]** What is the effect of different configuration parameters of GLM4MDE on the performance?
- **[RQ3]** What is the performance of using pretrained language models in GLM4MDE relative to training baseline ML methods from scratch to capture NL semantics?

In RQ1, we aim to investigate how pretrained language models and GNNs can be combined for conceptual modeling tasks. We perform node classification, edge classification, and link prediction through a self-supervised approach in which we mask (hide) each element's modeling-language semantics and train a model to recover them. We leverage existing labeled datasets for graph classification to frame a domain classification problem. By comparing three scenarios, namely, pure language model fine-tuning, structure-only GNN training, and a hybrid approach that stacks a GNN on top of a fine-tuned language model, we aim to determine which strategy delivers the best performance across all tasks and understand the effect of each strategy individually and combined.

In RQ2, we systematically assess how variations in key configuration parameters—specifically different data extraction strategies and the choice of k for neighborhood hops—affect overall performance. By examining these effects, we aim to demonstrate the importance of a framework that allows flexible, configurable data preparation to optimize downstream machine learning outcomes.

TABLE V
SUMMARY OF DATASET STATISTICS

DS	M	N	E	$\overline{N}$	$\tilde{N}$	$\overline{E}$	$\tilde{E}$	MinN	MinE	MaxN	MaxE	StdN	StdE
EA [26]	558	57910	70725	103	46	127	57	10	10	4003	5177	256.23	306.06
OnTo [74]	175	15890	20220	91	65	116	79	12	11	835	1304	96.60	140.37
Ecr [56]	444	18607	28786	42	25	65	35	5	10	662	1006	73.09	114.60
MS [73]	2539	119019	198110	47	23	78	33	4	10	428	1232	55.11	109.81

DS = Dataset, M = Models, N = Nodes, E = Edges
$\overline{N}/\overline{E}$ = mean nodes/edges per model; $\tilde{N}/\tilde{E}$ = median nodes/edges per model
MinN/MinE, MaxN/MaxE = minimum/maximum nodes or edges per model
StdN/StdE = standard deviation of nodes/edges per model
Dataset codes: EA = EAModelSet, OnTo = OntoUML, Ecr = Ecore-555, MS = ModelSet.

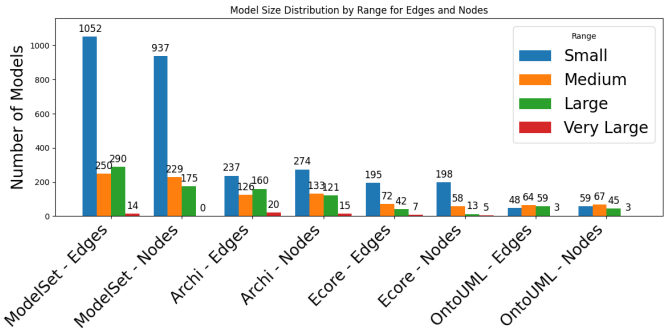


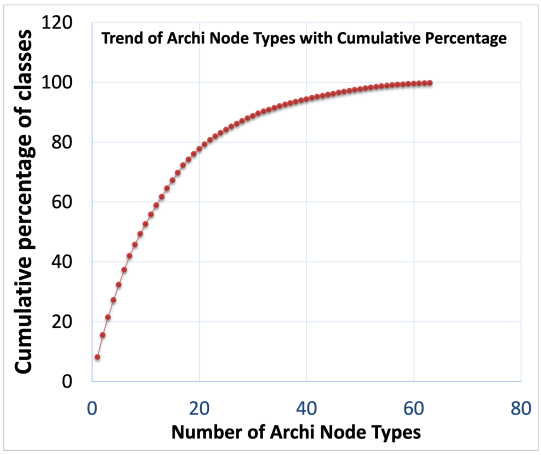
Fig. 10. Distributions of models based on model size

Finally, in RQ3, we evaluate the necessity of finetuning transformer-based language models to learn the information embedded in textual data, namely, natural language and modeling language semantics. We use existing simpler ML models, train them on the same data, and compare their performance with finetuned language models. We further introduce our deep learning model based on the transformer architecture, like BERT, and train that model from scratch for the node classification task. The response to this research question aims to answer whether i) finetuning an existing pretrained language models to adapt to the natural language and modeling language semantics benefits the performance on modeling tasks compared to using traditional ML models, ii) we can train transformer-based ML models from scratch to capture the natural language semantics instead of using pretrained language models.

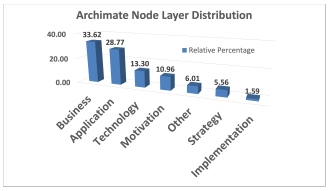
B. Datasets

We use 555 Ecore-555 [56] Ecore-based models, 4127 ModelSet [73] Ecore-based models, 175 OntoUML [74] models and 936 EAModelSet [26] ArchiMate models as datasets to conduct our evaluations. We exclude duplicate (i.e., exact duplicate match) models and all the models with edges $\leq$10 in the CKG representation of the model. Table V describes all four datasets we use for our evaluation. All the statistics in Table V are after filtering each dataset. The decision to exclude models with edges $\leq$10 is based on manual inspection. Furthermore, we can involve advanced filtering criteria to exclude undesired models for improved quality control over the models dataset using an existing model cleansing framework for conceptual models [71][13].

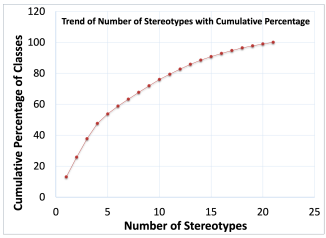
[13]https://github.com/junaidiiith/model-cleansing/



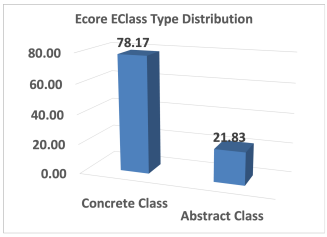
(a) Trend of ArchiMate Node Type Cumulative Percentage



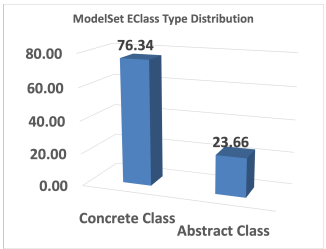
(b) ArchiMate Node Layer



(c) OntoUML Stereotype



(d) Ecore 555 EClass Type



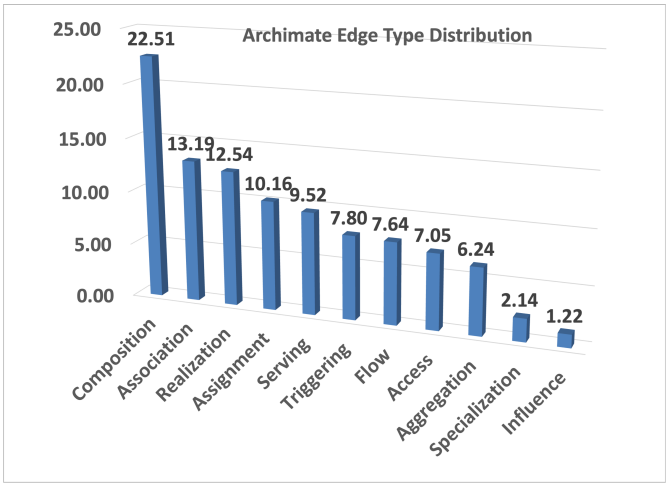
(e) ModelSet EClass Type

Fig. 11. Node Types Distribution in various datasets

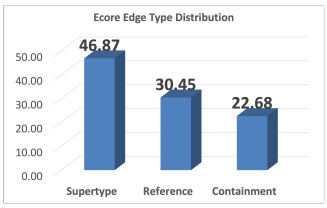
In Fig. 10, we show the share of different sizes of models by the number of nodes and edges. The legend in the figure provides the classification of models into various categories of model sizes. Based on the categorization of model size, the figure shows that models are predominantly small, followed by medium and large. There are very few models with more than 500 nodes. The figure shows that the distribution of the number of graphs by size is similar in all the datasets.

C. Experiments

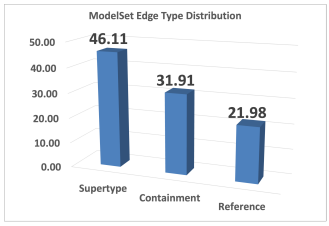
In order to respond to the proposed research questions, we ran five different experiments, each with several configurations. In the following, we describe each experiment. All nodes have a label; therefore, using the node label is the default information vectorizing a graph. However, not all cases apply to all datasets. In case of ArchiMate, we do not have attributes of nodes or edge labels, therefore we can only apply *use_node_label* (N_L), *use_node_type* (N_T),



(a) ArchiMate Edge Type

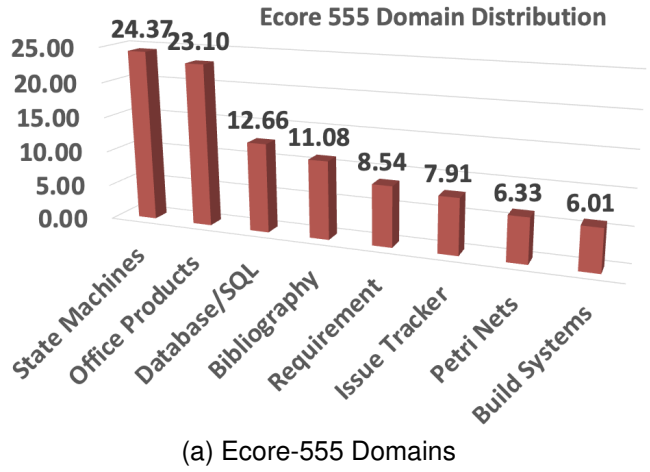


(b) Ecore 555 Edge Type

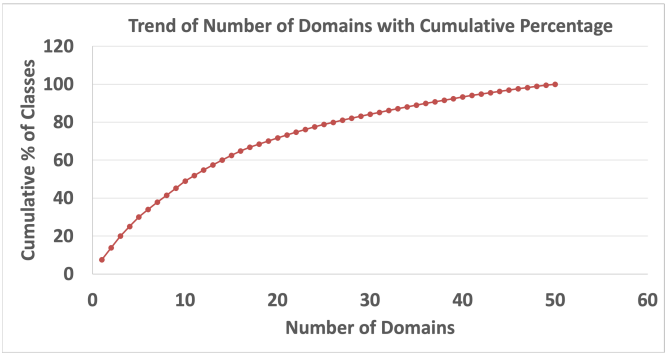


(c) ModelSet Edge Type

Fig. 12. Edge Types Distribution in various datasets



(a) Ecore-555 Domains



(b) ModelSet #Domains

Fig. 13. Graph Type Distribution in various datasets

and *use_edge_type* (E_T) to extract the model information for text vectorization and subsequent ML model training. In case of OntoUML, classes have attributes; therefore, we can apply *use_node_attributes* (N_A). Note that in OntoUML, apart from classes, relationships also have stereotype information; therefore, during the transformation of OntoUML models into a CKG, we transform the relationships into nodes and connect the source and destination of edges to the newly created node. This way, we can use node classification to predict missing classes and relationships in a partially completed OntoUML model. After the transformation of an OntoUML model into CKG, the *generalization* relationship is still considered an edge. After this transformation, the edges do not have a label as the newly created edge does not have a label, and OntoUML does not have a label for *generalization*. The edges of type *generalization* do have a *type*; however, given that the relationships, i.e., edges of type *Relation* are transformed into nodes, we do not apply *use_edge_type* (E_T) or *use_edge_label* (E_L) in the case of OntoUML. Finally, in the case of Ecore and ModelSet, we can use all the different configurations because both nodes and edges have types and labels. It is important to note that in the case of GNN configurations, in order to get the node and edge embeddings before GNN training, we vectorize the node and edge texts using the corresponding finetuned LLM that was used to extract text data using the same data extraction configuration parameters. For example, if *use_node_types* and *use_edge_label* were used to extract the data from the graphs and finetune an LLM, the finetuned LLM model that was used with the same configuration will be used to vectorize nodes and edges text for the GNN configuration that also has *use_node_types* and *use_edge_label* enabled for data extraction for consistency and to isolate the impact of the GNNs on performance. This experimentation choice not only allows for a meaningful comparison among the three approaches (isolated LLM finetuning, isolated GNN training, and combining both) but also prevents data leaks through a different LLM finetuned on a configuration different from that of the GNN configurations.

In the node data extraction phase (see Fig. 6), we use $k = set\{0, 1, 2, 3\}$ hops to get the data for each node. GLM4MDE is capable of supporting any GNN model supported by the PyTorch Geometric library[14], which is a standard library written using PyTorch[15] to easily write and train GNN for a wide range of applications related to structured data. However, for our experiments, we use GraphSAGE. In all the experiments, we consider 20% of elements, i.e., nodes for node classification, edges for edge classification and link prediction, and graphs for graph classification as test data and the remaining 80% as the training data. We remove the relevant information depending on the experiment from models and train an ML model on the remaining 80% of the nodes for all the datasets. Then we use the trained ML model on training nodes to predict the removed information.

[14]https://pytorch-geometric.readthedocs.io/en/latest/
[15]https://pytorch.org/

Experiment 1: In the first experiment, we aim to answer RQ1 and RQ2 using node classification, i.e., by predicting the modeling language semantics of CKG nodes. In case of the EAModelSet, given that two different semantic labels are present, i.e., node *type* and *layer*, we execute a node classification experiment for both labels separately. In the case of OntoUML, we aim to predict the node's stereotype label. In the case of Ecore and ModelSet, we aim to predict if a class is an abstract class or not. Node classification for EAModelSet and OntoUML involves multi-class classification, given that multiple categories of type, layer, and stereotype exist. In contrast, for Ecore and ModelSet, there is only one binary category, making this node classification a binary node classification task. In node classification, we can use the node type information of 80% of the training nodes while using the *use_node_type* configuration. However, it is crucial to note that for Ecore and ModelSet, using the edge type information of an *Abstract* class leaks the node type information into the training set because an abstract class is always connected with a supertype relationship. Therefore, we cannot use the *use_edge_type* configuration for node classification in the Ecore and ModelSet datasets. However, we can use the edge label information.

In this experiment, we first use different configurations for nodes' text classification with a language model, thereby getting a fine-tuned model. We then use this fine-tuned model to generate text embeddings for different nodes and edges, and then train a GNN model using these embeddings and the adjacency matrices of the graphs. We also train the GNN on randomized embeddings to only use the graph structural information in the adjacency matrix. This enables us to evaluate the role of structural information in isolation for the node classification task. In Fig. 11, we show the distribution of all the node types. The figure shows the uneven distribution of the classes in the dataset as the top 20 classes occur about 80% of the times and the remaining classes occur only 20% of the times. Fig. 11(b) shows the distribution of layer information in the nodes, and again we see an uneven distribution with Business layer occurring in over a third of nodes. Similarly, for the OntoUML dataset, we see that the top occurring classes already make up about 70% of the nodes. In the case of OntoUML, even though the total stereotypes we found in the dataset was 91. Upon manual inspection, we found this due to the inconsistent labeling of ontological stereotypes. Therefore, we consider only the stereotypes that occur at least 100 times, which makes a total of 21 stereotypes for the OntoUML dataset (see Fig. 11(c)).

Experiment 2: In the second experiment, we aim to answer RQ1 and RQ2 using edge classification, i.e., predicting the modeling language semantics of the graph edges. In particular, for all three datasets, i.e., EAModelSet, Ecore, and ModelSet, we aim to predict the type information of 20% of the edges for which we remove the edge type information before training the ML model. We do not execute this experiment for the OntoUML dataset because the relationships are modeled as nodes in the case of OntoUML. In this experiment, we first fine-tune the language model and then use the embeddings from the fine-tuned language model to train the GNN. We train the GNN using randomized embeddings to evaluate GNN performance using only structural information. In Fig. 12, we show the edge types' distribution. The percentage of the highest appearing class forms the baseline for that dataset, e.g., for EAModelSet, Composition occurs the highest, i.e., 22.51% in Fig. 12(a) and 46.11% for SuperType in ModelSet as shown in Fig. 12(c).

Experiment 3: In the third experiment, we aim to answer RQ1 and RQ2 by doing link prediction, i.e., predicting if a link should exist between two nodes or not. In this experiment, 20% of the test edges that are removed and 80% of the training edges that are preserved are considered as positive edges. In addition, the same percentage of test and train negative edges is also created using the existing edges, also termed positive edges in this case. A negative edge is an edge that has both of its endpoints in the graph, but the created edge is not part of the positive edges. Note that again, we do not execute this experiment for the OntoUML dataset. On running this experiment for various datasets, we found that GNNs could not learn how to predict the links, and further research needs to be done to investigate how to improve GNN performance for link prediction. Therefore, for this experiment, we use only the language modeling component of GLM4MDE. Nevertheless, GLM4MDE does support GNN training for link prediction as well. Given that this is a binary classification problem, i.e., given two nodes, should a link exist or not, the baseline for this task is 50% as we have half positive and half negative test edges.

Experiment 4: In this experiment, we aim to answer RQ1 and RQ2 by doing graph classification, i.e., predicting the domain of each model. In this experiment, we only use Ecore and ModelSet because these are the only two labeled datasets available. Again, we use randomized embeddings to predict the domain of the graph using only structural information with GNNs. In Fig. 13, we show the distribution of all the graph domain types. The percentage of the highest appearing class forms the baseline for that dataset, e.g., for ModelSet, the StateMachine class occurs the highest, i.e., 7.50% in Fig. 13(b) and 24.37% for State Machines in Ecore as shown in Fig. 13(a).

Experiment 5: In this experiment, we aim to answer RQ3 by doing a comparative evaluation of node classification performance by the language model component of GLM4MDE against i) simpler techniques, i.e., TF-IDF and word2vec with SVM or Random Forest, and ii) implementing a transformer-based neural network from scratch, i.e., the same architecture as BERT and training the neural network for node classification. We do this experiment on EAModelSet and OntoUML datasets because these two datasets have a higher complexity due to a higher number of classes than Ecore-555 and ModelSet, which have only two classes for node classification. The objective of this experiment is twofold. Firstly, it aims to assess the need for **pretrained** LLMs to capture the natural language semantics. Secondly, it compares

the expressive power of transformer-based techniques against simpler existing ML classifiers on our datasets. We train a relatively smaller transformer-based model from scratch for node classification.

TABLE VI
NODE CLASSIFICATION F1-SCORES RESULTS

Config	$k=0$	$k=1$	$k=2$	$k=3$
EAModelSet, Layer				
LLM(N_L)	0.742	0.826	0.817	0.820
LLM(N_L+N_T)	–	0.841	0.821	0.818
LLM(N_L+N_T+E_T)	–	**0.843**	0.835	0.834
GNN(R)	0.377	–	–	–
GNN(FTLM(N_L))	0.774	0.841	0.829	0.828
GNN(FTLM(N_L+N_T))	0.775	0.869	0.837	0.832
GNN(FTLM(N_L+N_T+E_T))	–	**0.879**	0.855	0.841
EAModelSet, Type				
LLM(N_L)	0.640	0.713	0.711	0.717
LLM(N_L+N_T)	–	0.728	0.719	0.718
LLM(N_L+N_T+E_T)	–	**0.761**	0.747	0.745
GNN(R)	0.164	–	–	–
GNN(FTLM(N_L))	0.659	0.692	0.679	0.691
GNN(FTLM(N_L+N_T))	–	0.730	0.712	0.720
GNN(FTLM(N_L+N_T+E_T))	–	**0.758**	0.671	0.731
OntoUML, Stereotype				
LLM(N_L)	0.502	0.708	0.726	0.728
LLM(N_L+N_A)	0.517	0.716	0.718	0.727
LLM(N_L+N_A+N_T)	–	0.752	0.753	**0.755**
GNN(R)	0.159	–	–	–
GNN(FTLM(N_L))	0.457	0.722	0.749	0.737
GNN(FTLM(N_L+N_A))	0.460	0.743	0.725	0.776
GNN(FTLM(N_L+N_A+N_T))	–	0.770	**0.855**	0.841
Ecore-555, Abstract				
LLM(N_L)	0.868	0.903	0.881	0.884
LLM(N_L+N_A)	0.892	0.905	0.897	0.893
LLM(N_L+N_A+N_T)	–	**0.914**	0.909	0.908
LLM(N_L+N_A+N_T+E_L)	–	0.909	0.896	0.851
GNN(R)	0.795	–	–	–
GNN(FTLM(N_L))	0.920	0.921	0.915	0.915
GNN(FTLM(N_L+N_A))	0.926	**0.930**	0.918	0.922
GNN(FTLM(N_L+N_A+N_T))	–	0.929	0.916	0.920
GNN(FTLM(N_L+N_A+N_T+E_L))	–	0.924	0.916	0.912
ModelSet, Abstract				
LLM(N_L)	0.842	0.885	0.891	0.877
LLM(N_L+N_A)	0.858	0.898	0.891	0.893
LLM(N_L+N_A+N_T)	–	0.893	0.896	0.890
LLM(N_L+N_A+N_T+E_L)	–	**0.900**	0.877	0.881
GNN(R)	0.824	–	–	–
GNN(FTLM(N_L))	0.829	0.827	0.829	0.829
GNN(FTLM(N_L+N_A))	0.897	0.910	0.909	0.889
GNN(FTLM(N_L+N_A+N_T))	–	**0.919**	0.910	0.904
GNN(FTLM(N_L+N_A+N_T+E_L))	–	0.905	0.903	0.905

N_L = use_node_label, N_A = use_node_attributes
N_T = use_node_type, E_L = use_edge_label
LLM(X) = LLM finetuning on text X.
FTLM(X) = finetuned language model producing embeddings for text X.
GNN(FTLM(X)) = GNN training on finetuned language model embeddings of text X.

Statistical Testing Methodology

To assess whether the observed performance differences across configurations are statistically reliable, we conducted paired **t-tests** and non-parametric **Wilcoxon signed-rank tests** [75] on the F_1 scores over 10 random seeds per configuration. The paired t-test assumes that (i) the paired differences between two methods are approximately normally distributed, and (ii) observations are independent across seeds. Both conditions are satisfied in our setting because each run uses an independent random seed. For robustness, we also employed the Wilcoxon signed-rank test, which does not assume normality. This combination ensures that our statistical claims (e.g., whether a GNN+FTLM configuration significantly outperforms an LLM-only baseline) are supported both under parametric and non-parametric assumptions, strengthening the reliability of our conclusions.

VI. RESULTS

In this section, we report the results of our experiments and respond to the proposed research questions.

A. Response to RQ1

In the following, we respond to RQ1 by providing performance details across the four types of tasks, namely, NC, EC, LP, and GC.

Node Classification: We evaluate the performance for each dataset and show the results of node classification in Table VI. In the EAModelSet nodes' *layer* label classification, we get the highest F1-score of 0.843 using one hop ($k=1$) and the data configuration that uses node and edge types apart from the node label. On further using GNNs for node classification, we get an increase in the F1-score with the highest score being 0.879 for $k=1$ hops. In the classification of node *type* label for the EAModelSet dataset, we get the highest F1-score of 0.761 again using the same configuration as in *layer* while finetuning the LLM. However, on further using GNNs with the finetuned embeddings, we get the highest performance of F1-score 0.755 in a different configuration ($k=3$, using node labels, types, and edge types information), although the difference is minimal, i.e., 0.752 for the same corresponding configuration by only finetuning an LLM. It is interesting to note that in the case of *type*, we do not get an increase in the performance using GNNs, instead getting a quite similar performance to that of an LLM, whereas in the case of *layer*, we did get an increase. This indicates that GNNs may not significantly improve the performance. However, given that the semantics captured in the node embeddings come from a language model finetuned on the same dataset, the performance will not significantly degrade using GNNs. Therefore, it is beneficial to try to use GNNs on top of the finetuned embedding in search for a further increase in performance. Furthermore, in general, the scores for *type* predictions are lower than those for *layer* predictions. This may be because of a much higher number of classes, i.e., 64 compared to eight in the case of *layer*.

In the OntoUML stereotype label prediction case, the language model finetuning gives the highest F1-score of 0.755. Combining GNNs with the finetuned language model (FTLM), we get a jump of up to 10 percentage points (pp) to 0.8551 F1-score. Therefore, in the case of OntoUML, there is a clear advantage in using GNNs on top of fine-tuned language model embeddings. Moreover, it is interesting to note that the highest

performance does not come with the same value of k. In the case of LM, we get the best performance at $k = 3$, whereas for GNN, we get the best performance at $k = 2$. In the case of Ecore-555 Abstract node label prediction, the scores are relatively high with a 0.914 F1-score. Using GNNs with FTLMs, we again get an increase in the performance, with 0.923 being the highest F1-score, using the same value of k but not the same data configuration. In ModelSet's case, we get similar results to Ecore-555, which is consistent with the fact that the models of both datasets have the same metamodel, i.e., Ecore. The classification results for ModelSet are also relatively high, with an F1-score of 0.9002, and augmenting GNNs gives a slight increase in performance, with an F1-score of 0.919.

We further note that using the R configuration, i.e., using GNNs with randomized embeddings, the structural information alone does not provide sufficient results for the case of EAModelSet or OntoUML with only about 0.16 F1-score for EAModelSet type and OntoUML stereotype and about 0.38 for EAModelSet layer. However, in the case of Ecore datasets, i.e., Ecore-555 and ModelSet, GNNs provide quite good results using only the node's structural information, classifying whether a class is an abstract class or not, with an F1-score of 0.7945 in the case of Ecore-555 and 0.824 for ModelSet. This may be due to certain structural features specific to abstract classes compared to concrete classes, e.g., the *degree* of abstract classes is higher than that of the concrete classes, as an abstract class is connected to several concrete classes. Further, Fig. 14 shows that LLM fine-tuning alone matches GNN-augmented performance in the ArchiMate node *type* classification (see Fig. 14(c)). In most other cases—including high-class-count datasets like OntoUML with 21 categories—adding a GNN consistently improves node-classification accuracy. From this result, we can conclude that augmenting GNN with LLM finetuning can benefit performance in the node classification task. However, using GNNs alone with structural information is always insufficient.

Edge Classification: We show the results of edge classification in Table.VII. In EAModelSet, FTLM provides the highest F1-score for $k = 2$ while using both node types and edge types of the training edges for the edge classification task involving 11 different types of edges. On augmenting the FTLM with GNN training on finetuned node embeddings, we got a jump in edge classification performance of over 11 pp to 0.941 F1-score. Interestingly, using only the node label embeddings from the finetuned language model was sufficient for GNNs to predict the type of edges with a high performance of 0.941 F1-score. In the case of Ecore and ModelSet, FTLMs alone provide high performance with an F1-score of 0.944 for Ecore and 0.948 for ModelSet. Augmenting GNNs with LLM finetuning improves the performance by almost 4 pp in both Ecore and ModelSet. This similarity in behaviour can again be attributed to the fact that both datasets have a difference of only the number of models and model size. From the modeling language semantics point of view, both datasets have the same metamodel, so structurally they will overlap.

TABLE VII
EDGE CLASSIFICATION F1-SCORES RESULTS

Config	$k=0$	$k=1$	$k=2$	$k=3$
EAModelSet				
LLM(N_L)	0.778	0.815	0.832	0.834
LLM(N_L+N_T)	0.794	0.806	0.827	0.833
LLM(N_L+N_T+E_T)	–	0.829	**0.835**	0.829
GNN(R)	0.492	–	–	–
GNN(FTLM(N_L))	0.559	0.837	**0.941**	0.900
GNN(FTLM(N_L+N_T))	0.934	0.907	0.746	0.749
GNN(FTLM(N_L+N_T+E_T))	–	0.876	0.920	0.937
Ecore-555				
LLM(N_L)	0.907	0.921	0.930	0.933
LLM(N_L+N_A)	0.917	0.919	0.855	0.929
LLM(N_L+N_A+N_T)	0.920	0.903	0.935	0.927
LLM(N_L+N_A+N_T+E_L)	–	0.930	**0.944**	0.801
LLM(N_L+N_A+N_T+E_L+E_T)	–	0.921	0.915	0.828
GNN(R)	0.685	–	–	–
GNN(FTLM(N_L))	0.968	0.971	0.979	0.840
GNN(FTLM(N_L+N_A))	0.969	0.974	0.933	0.785
GNN(FTLM(N_L+N_A+N_T))	0.975	0.973	0.984	0.816
GNN(FTLM(N_L+N_A+N_T+E_L))	–	0.984	**0.987**	0.844
GNN(FTLM(N_L+N_A+N_T+E_L+E_T))	–	0.968	0.969	0.981
ModelSet				
LLM(N_L)	0.908	0.934	0.948	0.932
LLM(N_L+N_A)	0.917	0.937	0.945	0.936
LLM(N_L+N_A+N_T)	0.916	0.935	0.891	0.932
LLM(N_L+N_A+N_T+E_L)	–	0.940	0.929	**0.948**
LLM(N_L+N_A+N_T+E_L+E_T)	–	0.941	0.884	0.942
GNN(R)	0.667	–	–	–
GNN(FTLM(N_L))	0.967	0.980	0.978	0.823
GNN(FTLM(N_L+N_A))	0.972	0.982	0.946	0.924
GNN(FTLM(N_L+N_A+N_T))	0.970	0.978	0.957	0.918
GNN(FTLM(N_L+N_A+N_T+E_L))	–	0.983	**0.986**	0.986
GNN(FTLM(N_L+N_A+N_T+E_L+E_T))	–	0.977	0.985	0.980

N_L = use_node_label, N_A = use_node_attributes
N_T = use_node_type, E_L = use_edge_label, E_T = use_edge_type
LLM(X) = LLM finetuning on text X.
FTLM(X) = finetuned language model producing embeddings for text X.
GNN(FTLM(X)) = GNN trained on embeddings from a finetuned language model.

In the case of edge classification, we evaluated the impact of augmenting GNNs on top of LLM finetuning. We see that in Fig. 15, the highest point comes from augmenting GNNs with LLM finetuning; however, the improvement with GNN augmentation is not consistent for different values of k. In case of ArchiMate *type*, for the GNN configuration that gives the highest F1-score 0.934 for $k = 0$, i.e., using node and edge types apart from node labels, has the lowest F1-score for $k = 3$. However, overall, the trend is still the same that the GNN augmentation is beneficial for overall performance as the GNN lines are above the LLM finetuning lines in Fig. 15(a). In the case of ModelSet, it is quite clear that GNN augmentation benefits performance; however, it is up to $k = 2$. For $k = 3$, the GNN performance drops significantly. The corresponding LLM finetuning also drops from $k = 2 \rightarrow 3$, but the drop in GNN performance is higher. This may be due to the inclusion of more homogeneous information for all nodes due to higher common neighbors.

Even though GNNs perform poorly using "random embeddings" for ArchiMate, they provide an F1-score of almost 0.5. In the case of Ecore and ModelSet, the performance is much better and reaches up to 0.67 F1-score.

This again indicates that the uninitialized structural features capture information for edge classification, but they alone are insufficient.

TABLE VIII
LINK PREDICTION F1-SCORES RESULTS

Dataset	Config	$k=0$	$k=1$	$k=2$	$k=3$
EAModelSet	N_L	0.771	0.807	0.826	0.828
Type	N_L+N_T	0.786	0.799	0.827	**0.830**
	N_L	0.907	0.921	0.929	0.931
Ecore-555	N_L+N_A	0.916	0.919	0.855	0.930
Type	N_L+N_A+N_T	0.920	0.904	**0.935**	0.927
	N_L	0.908	0.933	**0.948**	0.930
ModelSet	N_L+N_A	0.917	0.937	0.944	0.936
Type	N_L+N_A+N_T	0.916	0.934	0.890	0.932

N_L = use_node_label, N_A = use_node_attributes
N_T = use_node_type

Link Prediction: Next, we show the link prediction results in TableVIII. As mentioned previously, we do not include GNN finetuning for link prediction because GNNs provide performance similar to random initialization for the link prediction task, thereby not learning the patterns required to predict the presence of a link between two nodes accurately. Therefore, we relied on using only the language models and the k-hop structural information for the link prediction task.

In EAModelSet link prediction, the F1-score rises steadily from $k = 0$ to $k = 2$, then plateaus at $k = 3$. The best link-prediction F1-score is 0.8298 ($k = 3$, N_L+N_T). This mirrors edge classification because a 2-hop context captures the most relevant connectivity information for predicting missing links. Ecore-555 link prediction results show that N_T at $k = 2$ yields the best F1-score of 0.935, compared to 0.929 for node-label only. N_A (using node labels and their attributes) performs poorly at $k = 2$ (F1-score = 0.8546), indicating that low-level attribute text can mislead link inference when over-aggregated. In case of ModelSet, the peak is F1-score = 0.9475 (node-label only, $k = 2$). Unlike edge classification, here, additional node metadata yields only marginal gains—structural co-occurrence patterns in the 2-hop neighborhood dominate link-predictive power.

Graph Classification: In this task, we performed supervised domain classification. Table.IX shows that, for both Ecore-555 and ModelSet, the simplest "LM only" configuration using only node labels (N_L) already achieves strong performance (f1 $\geq$ 0.749 on ModelSet, $\geq$ 0.949 on Ecore-555). Adding node attributes (N_A) yields an increase on ModelSet (from 0.749 $\rightarrow$ 0.790 at $k = 0$) but slightly decreases on Ecore-555 at $k = 1$ (0.986 $\rightarrow$ 0.935). Introducing node types (N_T) on top of attributes does not impact ModelSet at $k = 1$ (0.820$\rightarrow$0.819) but negatively impacts Ecore-555 at higher hops. Including edge labels (E_L) and edge types (E_T) yields mixed effects: on ModelSet, the fullest language model configuration (N_L+N_A+N_T+E_L+E_T) recovers to 0.806 at $k = 2$ and 0.788 at $k = 3$, while on Ecore-555 it drops steadily to 0.873 at $k = 3$. In all cases, the pure LM scores are highest at $k = 1$ or $k = 0$, suggesting that immediate neighborhood information is most valuable when using only text-derived embeddings.

TABLE IX
GRAPH CLASSIFICATION F1-SCORES RESULTS

Config	$k=0$	$k=1$	$k=2$	$k=3$
Ecore-555				
LLM(N_L)	0.949	**0.986**	0.978	**0.986**
LLM(N_L+N_A)	0.978	0.935	0.934	0.957
LLM(N_L+N_A+N_T)	0.919	0.972	0.874	0.931
LLM(N_L+N_A+N_T+E_L)	-	0.888	0.874	0.936
LLM(N_L+N_A+N_T+E_L+E_T)	-	0.858	0.851	0.873
GNN(R)	0.284	0.275	0.284	0.275
GNN(FTLM(N_L))	0.871	0.935	0.962	0.968
GNN(FTLM(N_L+N_A))	0.978	**0.986**	0.974	**0.986**
GNN(FTLM(N_L+N_A+N_T))	0.978	0.957	**0.986**	0.943
GNN(FTLM(N_L+N_A+N_T+E_L))	-	0.908	0.950	**0.986**
GNN(FTLM(N_L+N_A+N_T+E_L+E_T))	-	0.892	0.959	0.939
ModelSet				
LLM(N_L)	0.749	0.787	0.780	0.776
LLM(N_L+N_A)	0.790	**0.820**	0.793	0.746
LLM(N_L+N_A+N_T)	0.798	**0.819**	0.779	0.750
LLM(N_L+N_A+N_T+E_L)	-	0.810	0.773	0.780
LLM(N_L+N_A+N_T+E_L+E_T)	-	0.809	0.806	0.788
GNN(R)	0.050	0.058	0.066	0.057
GNN(FTLM(N_L))	0.795	0.779	0.755	0.767
GNN(FTLM(N_L+N_A))	0.761	0.818	0.787	0.767
GNN(FTLM(N_L+N_A+N_T))	0.767	**0.827**	0.746	0.750
GNN(FTLM(N_L+N_A+N_T+E_L))	-	0.798	0.786	0.789
GNN(FTLM(N_L+N_A+N_T+E_L+E_T))	-	0.807	0.771	0.749

N_L = use_node_label, N_A = use_node_attributes
N_T = use_node_type, E_L = use_edge_label
E_T = use_edge_type, LLM(X) = LLM finetuning on text X.
FTLM(X) = finetuned language model producing embeddings for text X.
GNN(FTLM(X)) = GNN trained on embeddings from a finetuned language model.

Augmenting with a GNN ("+GNN" rows) increases the graph classification performance in a few cases. A simple randomized embedding baseline (R) without text features performs poorly (e.g., F1-score$\approx$0.284 on Ecore-555 and $\approx$0.058 on ModelSet at $k = 1$). However, once the finetuned text embeddings for N_L are injected, GNNs close the gap quickly: on Ecore-555, they reach 0.962$\rightarrow$0.968 at $k \geq 1$, and on ModelSet, they jump to 0.779$\rightarrow$0.827. More richly annotated configurations—FTLM(N_L+N_A) and beyond provide Ecore-555 with performance up to 0.986 at multiple hop-counts, and ModelSet up to 0.827 at $k = 1$. Notably, in the case of Ecore-555, GNN input (FTLM(N_L+N_A+N_T+E_L+E_T)) benefits from structural propagation; it rises from 0.892 at $k = 0$ to 0.939 at $k = 2$, but on ModelSet, it decreases from 0.807 at $k = 1$ to 0.771 at $k = 2$. Thus, combining text-based embeddings with graph convolution consistently outperforms modality alone, particularly when leveraging one- or two-hop neighborhood information.

Significance Testing for RQ1 We tested whether augmenting finetuned language models with GNNs (**FTLM+GNN**) yields statistically reliable gains over **FTLM** across tasks, datasets, k-hops, and text configurations (10 seeds each; paired t-test and Wilcoxon on per-seed F_1). In *EAModelSet edge classification* at k=1 with NL+NT, mean F_1 rises from 0.808 (FTLM) to 0.907 (FTLM+GNN), $\Delta = +0.099$, t=21.51,

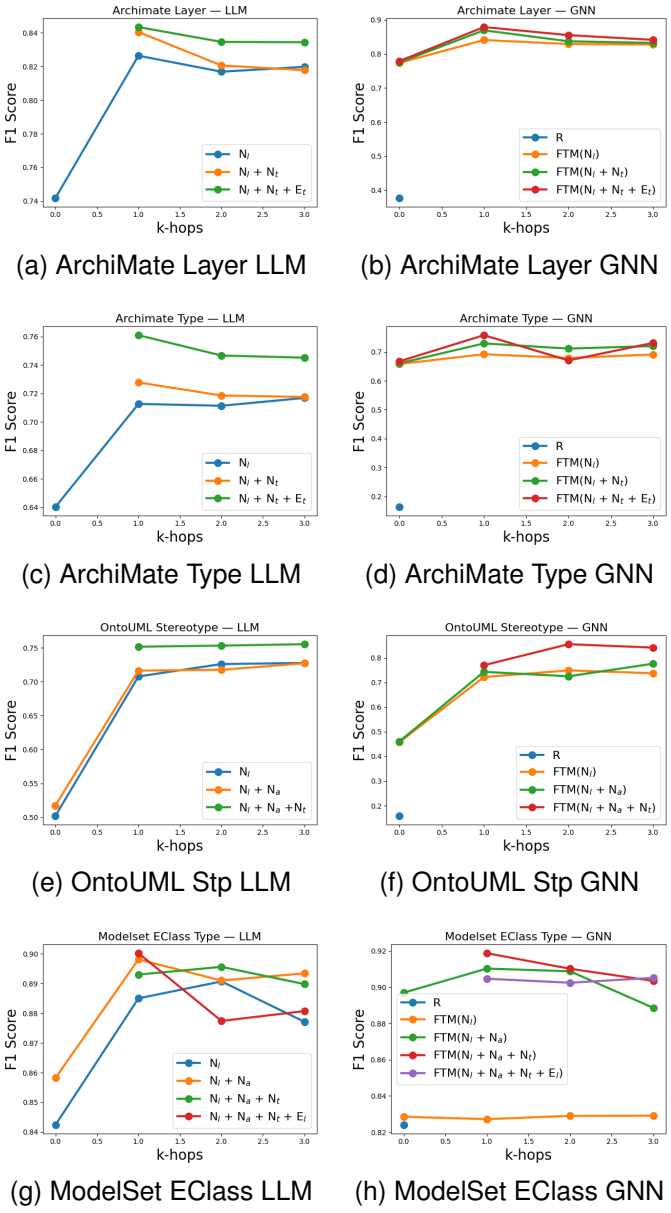


Fig. 14. Effect of data configs and k on GLM4MDE-based Node Classification

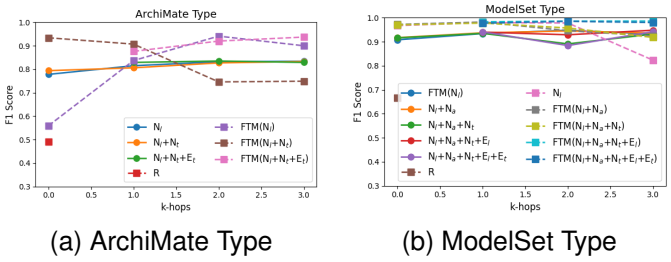


Fig. 15. Effect of data configs and k on GLM4MDE-based Edge Classification

$p<5\times10^{-9}$; Wilcoxon $p=0.002$; $d=6.8$. In *Ecore-555 edge classification* at $k=2$ with NL+NA+NT+EL, FTLM improves from 0.944 to 0.987 under FTLM+GNN ($\Delta=+0.043$, $t=6.91$, $p=3.4\times10^{-5}$). Even small deltas (e.g., $+0.021$ for EAModelSet NL at $k=2$) are significant ($t=4.38$, $p=1.8\times10^{-3}$; Wilcoxon $p<0.01$). Across all RQ1 comparisons (142 rows), **123/142** are significant under both tests; **102/142** improve with FTLM+GNN (median $\Delta=+0.021$), while **40/142** degrade (often at high k). Random-embedding GNN controls remain far below FTLM or FTLM+GNN, confirming that structure alone is insufficient without semantic grounding.

RQ1 Conclusion. Across modeling tasks, stacking a graph neural network on top of a finetuned language model reliably improves over language modeling alone, while structure without semantic grounding rarely suffices. The hybrid approach is therefore the recommended default for node and edge prediction, with pure language modeling retained only when graph signal is weak or noisy.

B. Response to RQ2

In the following, we discuss the effect of k-hops and the different configuration parameters on our experiments.

Effect of k**:** This configuration parameter is responsible for capturing structural information in the textual information of the node itself. A higher value of k indicates more information from neighbors. Therefore, we see in Fig. 14 that increasing k from 0 to 1 always increases the performance, and in the case of OntoUML, the change is quite significant, of more than 25%. However, the performance does not monotonically increase with k. The reason for this is that with higher values of k, the difference in the information for a given node reduces because of more common neighbors. Therefore, the ML model cannot find a distinguishing signal to classify the node into the correct category.

However, a question arises: to capture structural information in the graph, why do we use k-hop information if we already use GNNs, or why do we use GNNs if we are already using k-hop information? Does this not lead to redundant information? The reason is that "k-hop" text augmentation and a GNN inject structural (i.e., neighborhood) information into the node classifier. However, both ways do it in fundamentally different, and ultimately complementary, ways.

In case of k-hop text augmentation, we pull in the raw text of all nodes up to k-hops away and append it (or otherwise embed it) alongside a given node's text. This captures a fixed window of neighboring nodes' information, i.e., a bag of text from neighbors. However, the limitations of using such information is that i) this information is static, unweighted, i.e., every neighbor's information counts equally, regardless of the "importance" that neighbor, ii) this does not capture graph topology beyond distance, i.e., we may know that A, B and C are within k-hops of our current node T, but this does not tell us whether A–B is strongly connected, part of a clique, a bridge and so on and finally, iii) as k grows your neighbor texts overlap more, washing out the individual signal, which is consistent with the results as increasing k does not always give higher performance.

In case of GNN message-passing, we start from each node's embedding (already carrying its text + any k-hop context) and then *learn*—via multiple neural layers—to pull in, weight, and transform information from exactly those neighbors to which it is connected. This leads to i) learnable aggregation, i.e., the model figures out which neighbors (and which neighbor features) matter most for classification, ii) topology-aware propagation, i.e., edges, multi-hop connectivity patterns (e.g. hubs, bridges) and even the "shape" of the local graph (e.g. is this node on the perimeter of a community or at its core?) can all influence the final information captured by a node, and finally iii) stacking GNN layers allows mixing information over arbitrarily many hops without resorting to appending ever more raw text. This explains the difference between using k hops and GNNs separately and their value.

Effect of Data Configurations: We show the effect of data configuration for node classification in Fig. 14 and edge classification in Fig. 15. In EAModelSet for node's layer classification (see Fig. 14(a)), we see that using only the node's labels (N_L) provides similar performance to using node types information of the training nodes (N_T) for $k \geq 2$. For $k = 1$, using node types improves the F1-score from 0.826 to 0.84. However, using edge types (E_T) provided the best performance for all the tasks. This result makes sense because the type of edge connected to a node can reduce the search space to find the accurate class of a node. The trend in the results is quite similar for ArchiMate type (see Fig. 15(a)) as well, although the difference in N_T and E_T configuration performance is higher in the case of ArchiMate Type. This again can be explained by the relationship between edge type and node type. In layer classification, the number of classes is only 7; therefore, adding edge type information does not improve performance as much as it does in predicting the node type when the number of classes is much higher, i.e., 64. In case of OntoUML stereotype classification (see Fig. 14(e)) there is a configuration (N_L+N_A), i.e., using node attributes. Adding node attributes has a marginal improvement on the performance up to $k = 1$ but has a negative effect for $k = 2, 3$. However, using node types improves the performance of the node classification. In case of ModelSet (see Fig. 14(g)), the results are less orderly. Using attributes improves the performance for $k = 1$ from $k = 0$; however, the performance drops for $k = 2$ and increases again for $k = 3$. Using node types provides the highest performance for $k = 2$; however, the performance is not monotonic. Using edge types provides the highest performance for $k = 1$; however, it drops below every other configuration for $k = 2$ and then slightly increases for $k = 3$. Overall, using the different types of information captured within a model affects the performance, and no clear and consistent pattern or rule can be inferred as fixed to get a higher performance. This result empirically underpins the requirement to make the information provided to the ML model for training configurable according to the modeling language of choice.

Next, we provide a summary of how the different data-configuration settings work. LLM baseline ($N_L \rightarrow E_T$): Moving from just node-labels (N_L) to include node types (N_T) gives a small but consistent bump ($\approx$ 0.02 F1-score at k=0). Incorporating edge labels/types (E_L/E_T) yields the largest LM gains—especially at low k—because edge semantics prune the candidate classes more sharply. Using Random embeddings (R) fails outright. On augmenting GNNs with the finetuned embeddings of the node texts, we see that even with only node-text embeddings (FTLM(N_T)), adding GNN message-passing jumps F1-score into the high-0.8s/low-0.9s. Best results come when the LLM is fine-tuned on both node and edge text (FTLM(E_L/E_T)), pushing F1-score to $\geq$0.98 on Ecore-555 and ModelSet. Adding node types (N_T) brings slight gains in some settings (e.g., +0.015 at $k = 0$ on EAModelSet, +0.006 at $k = 2$ on Ecore-555), but is not universally better—on ModelSet, the plain N_L provides best scores at the optimal k. Link prediction is less sensitive to extra node/edge attributes than edge classification.

We note that edge classification demands richer textual context (edge labels/types) and benefits dramatically from combining FTLM embeddings with GNN message-passing. Link prediction, by contrast, relies more on the structure captured by k-hop augmentation; additional semantic features yield only marginal or inconsistent improvements.

Significance Testing for RQ2 (Effect of k) We quantified the effect of expanding neighborhood context (k) via paired tests between adjacent hops ($k: 0 \rightarrow 1$, $1 \rightarrow 2$, $2 \rightarrow 3$) for both LMs and FTLM+GNN. Across **205** hop-comparisons, **125** are significant (both tests). Gains and drops are nearly balanced overall (**98** gains vs. **107** drops), but the pattern is directional: $0 \rightarrow 1$ **almost always helps** (**38** significant gains), $1 \rightarrow 2$ **helps less often** (**24** gains), and $2 \rightarrow 3$ **frequently hurts** (**44** significant drops). The strongest gain appears in *EAModelSet EdgeCls* (FTLM+GNN, NL): k=0 $\rightarrow$ 1 lifts $0.558 \rightarrow 0.834$ ($\Delta = +0.277$, t=92.86, $p < 10^{-14}$). Non-monotonicity is clear: in the same task (FTLM+GNN, NL), k=1 $\rightarrow$ 2 adds $+0.110$ (t=26.97, $p \ll 10^{-8}$), but k=2 $\rightarrow$ 3 drops -0.039 (t=-9.55, $p < 10^{-5}$). Similar significant declines at $2 \rightarrow 3$ occur in Ecore-555 and ModelSet edge classification, particularly for rich configs (e.g., Ecore-555 LLM NL+NA+NT+EL: $0.941 \rightarrow 0.798$, $\Delta = -0.144$, $p < 10^{-3}$).

Significance Testing for RQ2 (Effect of data configuration) We tested pairwise "data configuration ladder" changes (e.g., NL$\rightarrow$NL+NT, NL+NA+NT$\rightarrow$+EL, ...) at fixed k and model type. Across **110** comparisons, **71** are significant (both tests); effects are **dataset-dependent** (53 positive vs. 57 negative significant shifts). Notable gains include *Ecore-555 EdgeCls* (FTLM+GNN, k=3) moving from NL+NA+NT+EL to NL+NA+NT+EL+ET: $0.846 \rightarrow 0.976$ ($\Delta = +0.130$, t=35.91, $p \ll 10^{-10}$). In contrast, *EAModelSet EdgeCls* (FTLM+GNN, k=3) NL+NT $\rightarrow$ NL+NT+ET *reduces* $0.749 \rightarrow 0.668$ ($\Delta = -0.082$, t=-18.83, $p < 10^{-8}$). For graph classification, adding NT can hurt on Ecore-555 LLM at k=0 (NL+NA 0.979 $\rightarrow$ NL+NA+NT 0.918, $\Delta = -0.061$, t=-13.04, $p < 10^{-6}$). These results confirm that edge semantics (EL/ET) often boost edge- and node-level tasks but can overfit or inject noise at higher k, while NT/NA effects vary by dataset and task.

In summary, as a response to RQ1, we note that GNNs on finetuned LM embeddings outperform using only language

model finetuning by 4–10 percentage points across all four datasets. Even though using GNN with "Random embeddings" (R) learn some patterns, they produce very low F1-score, showing that structural information must be grounded in meaningful text embeddings.

In response to RQ2, moving from $k = 0 \rightarrow 1$ always yields an F1-score increase, since even one-hop context adds crucial edge-neighborhood signals. The optimum k varies by dataset and config (often $k = 2$), and performance sometimes degrades at $k = 3$—likely because overly broad neighborhoods introduce noise and dilute the discriminative edge features. As a further response to RQ2, among LM-only runs, appending edge labels or types consistently outperforms using only node labels or types, especially when the number of edge classes is large (e.g., EAModelSet, Type). When GNNs are added, the *use_edge_types* configuration remains the strongest in most cases. This suggests edge-level metadata is the richest signal for this task, and GNN message-passing can most effectively leverage it when explicitly supplied.

RQ2 Conclusion. Expanding neighborhood context to one hop is consistently beneficial; extending to two hops helps in some settings; pushing to larger neighborhoods often introduces noise and harms accuracy. Adding edge semantics and types is frequently valuable at modest neighborhood sizes, but rich configurations at large neighborhoods can overfit. Configuration should thus be tuned per dataset with restrained hops and selectively added features.

C. Response to RQ3

Given that most existing works use textual labels for ML4MDE tasks, in the following, we compared the performance of the language model finetuning component of our framework with the existing simpler techniques, modeling domain-specific word embeddings [76] to capture the semantics of the model text. We provide an in-depth analysis of how each family of models behaves across our node classification tasks, i.e., layer and type on EAModelSet, stereotype on OntoUML, abstract on Ecore-555, and abstract on ModelSet. We consider TF-IDF + SVM →Worde4MDE + Random Forest as the classification baselines. We consider CMGPT as a "from-scratch transformer" and finally, our fine-tuned language model (FTLM, GLM4MDE).

Classical Baselines: Using classical models, we see modest gains from $0 \rightarrow 1$ hops, then we see a plateau in F1-scores. Across nearly every dataset, moving from $k = 1 \rightarrow 2$ gives a small increase (often +0.02–0.10 F1-score). This result matches the intuition that seeing a neighbor's label helps disambiguate a node's class. Beyond $k = 1$, improvements stop or even reverse. These shallow feature pipelines cannot pick and choose among dozens of neighboring phrases, so they treat all appended words equally, adding noise to larger neighborhoods. TF-IDF and word2vec, GloVE, Worde4MDE provide static embeddings that do not change with context and therefore struggle with domain-specific vocabulary. GloVe is the weakest across the board (e.g., 0.353–0.410 on EAModelSet layer, vs. 0.635–0.677 for TF–IDF). GloVe's general-domain vectors were not trained on modeling-language text ("Actor," "Invoice," "Aggregation"), so many key tokens end up with near-random embeddings. Switching to Worde4MDE, which was pretrained on software-and-modeling corpora, yields a noticeable boost (e.g., on Ecore-555, Worde4MDE at $k = 1$ is 0.800 vs. GloVE's 0.441). That underscores the value of domain adaptation even for static embeddings. Furthermore, we see that Random Forest can sometimes outperform SVM. For example, on tasks with more classes (e.g., EAModelSet has 64 type classes), GloVE/Worde4MDE (RF) slightly outperforms GloVE/Worde4MDE (SVM) at $k = 2$. We see that the existing methods involving shallow pipelines, i.e., not using deep learning models, can provide an F1-score of about a

TABLE X
LANGUAGE MODEL CLASSIFICATION F1-SCORES RESULTS

ML Model	$k=0$	$k=1$	$k=2$	$k=3$
EAModelSet, Layer				
TF-IDF + SVM	0.635	0.681	0.673	0.677
GloVe + SVM	0.353	0.357	0.410	0.410
GloVe + Random Forest	0.598	0.601	0.624	0.632
Worde4MDE + SVM	0.424	0.434	0.527	0.520
Worde4MDE + Random Forest	0.588	0.590	0.618	0.625
CMGPT	0.659	0.676	0.679	0.678
GLM4MDE Finetuned LM	0.742	**0.843**	0.835	0.834
EAModelSet, Type				
TF-IDF + SVM	0.539	0.536	0.531	0.526
GloVe + SVM	0.360	0.326	0.335	0.330
GloVe + Random Forest	0.507	0.485	0.499	0.498
Worde4MDE + SVM	0.428	0.419	0.424	0.415
Worde4MDE + Random Forest	0.494	0.478	0.489	0.487
CMGPT	0.506	0.526	0.525	0.528
GLM4MDE Finetuned LM	0.640	**0.761**	0.747	0.745
OntoUML, Stereotype				
TF-IDF + SVM	0.179	0.185	0.190	0.177
GloVe + SVM	0.341	0.525	0.478	0.434
GloVe + Random Forest	0.382	0.516	0.489	0.474
Worde4MDE + SVM	0.386	0.596	0.541	0.528
Worde4MDE + Random Forest	0.377	0.537	0.524	0.499
CMGPT	0.295	0.405	0.402	0.402
GLM4MDE Finetuned LM	0.517	0.752	0.753	**0.755**
Ecore-555, Abstract				
TF-IDF + SVM	0.745	0.748	0.706	0.635
GloVe + SVM	0.441	0.441	0.441	0.441
GloVe + Random Forest	0.698	0.800	0.789	0.762
Worde4MDE + SVM	0.606	0.548	0.468	0.482
Worde4MDE + Random Forest	0.698	0.788	0.778	0.745
CMGPT	0.824	0.821	0.822	0.823
GLM4MDE Finetuned LM	0.892	**0.914**	0.909	0.908
ModelSet, Abstract				
TF-IDF + SVM	0.690	0.693	0.682	0.650
GloVe + SVM	0.000	0.000	0.000	0.000
GloVe + Random Forest	0.443	0.605	0.597	0.584
Worde4MDE + SVM	0.000	0.000	0.000	0.000
Worde4MDE + Random Forest	0.453	0.580	0.589	0.570
CMGPT	0.792	0.791	0.792	0.791
GLM4MDE Finetuned LM	0.858	**0.900**	0.877	0.881

mid-0.5–0.7 range with the right domain embeddings and one-hop context, but they plateau quickly. Domain-pretrained static embeddings, i.e., (Worde4MDE), are better than off-the-shelf GloVe—but still far below using a finetuned transformer-based deep learning model. This indicates that embeddings should be at minimum pretrained (or adapted) on in-domain text for any new modeling language.

CMGPT from Scratch: A transformer-based classifier trained from random initialization, with the same input text as the LM baselines. We observe quite a bit of improvement over classical baselines, e.g. on EAModelSet layer, CMGPT jumps to 0.659→0.679 F_1 (vs. 0.353–0.635 for GloVe/SVM, Worde4MDE/SVM). It clearly learns useful word order and phrase structure patterns that bag-of-words cannot. However, we see that CMGPT is still well below finetuned LM. CMGPT's peak (0.679 at $k = 2$) is $\leq$15 points under FTLM's 0.843 at $k = 1$. This gap persists across all five tasks. Finally, the effect of k on CMGPT's performance is only modest. The performance improves up to $k = 2$, then flattens or decreases, similar to the classical pipelines. Therefore, CMGPT cannot learn to weight distant neighbors appropriately without large-scale pretraining. CMGPT must learn all semantic representations—entity classes, edge labels, modeling-language jargon—from scratch on rather small classification datasets (a few thousand examples). That leads to limited vocabulary coverage and brittle performance. Therefore, we note that training a transformer from zero does outperform classical methods. However, it demands far more labeled data (or data augmentation) to match the performance of a finetuned LM. In the absence of hundreds of thousands of in-domain sentences for pretraining, which is likely the case for ML4MDE research where the large and good quality of datasets are lacking, CMGPT-style training will likely plateau in the mid-0.6–0.7 F_1 range for such a classification task.

FTLM: In this case, we use a general-purpose pretrained transformer (e.g. BERT-style) that we further finetune on each modeling-language classification task, using the fullest config. We observe a massive jump at one hop. F_1 jumps by +0.08–0.11 on every dataset going from $k = 0 \rightarrow 1$. This one-hop context appears to be enough neighbor semantics to disambiguate classes without overwhelming the model with redundant text. Peak performance outstrips every other ML model by a significant margin. In case of layer classification of EAModelSet, 0.843 at $k = 1$ vs. 0.681 (TF-IDF + SVM) or 0.676 (CMGPT). In the stereotype classification of OntoUML, we get 0.752 at $k = 1$ for FTLM vs. 0.596 (best baseline). Finally, in the case of FTLM, we see only minor drops beyond $k = 1$, showing a slight robustness to adding more hops—performance decreases by < 0.01 after $k = 2$ because its self-attention can learn to ignore or down-weight repetitious neighbor text. Therefore, pretraining on massive general-domain corpora gives the model an understanding of English semantics, syntax, and graph-related language patterns. Secondly, exposing the LLM to several thousand examples of "Actor" vs. "Entity" vs. "Boundary"—and to phrases like "Customer places Order" vs. "Customer reviews Order"—enables it to specialize its embeddings to our modeling-language ontology. Finally, when we append neighbor labels (E_T) or edge labels (E_L), the transformer's self-attention can learn which tokens matter most for this particular node or this specific edge, avoiding the over-aggregation problem classical methods suffer.

Significance Testing for RQ3 We compared FTLM against TF-IDF+SVM, GloVe (SVM/RF), Worde4MDE (SVM/RF), and a from-scratch transformer (CMGPT) over 10 seeds. All FTLM vs. baseline pairs are significant (both tests). Extremes: *ModelSet Abstract* at k=1 vs. Worde4MDE+SVM jumps $0.006 \rightarrow 0.901$ ($\Delta = +0.895$, t=292.89, $p<10^{-18}$); vs. GloVe+SVM: $0.000 \rightarrow 0.901$ ($\Delta = +0.901$, t=1164.8, $p \ll 10^{-30}$). On *EAModelSet Layer* at k=1, TF-IDF+SVM $0.678 \rightarrow$ FTLM 0.839 ($\Delta = +0.161$, t=24.48, $p<10^{-9}$), and GloVe+SVM $0.358 \rightarrow 0.839$ ($\Delta = +0.481$, t=85.36, $p<10^{-13}$). For *OntoUML Stereotype* at k=1, best baseline (Worde4MDE+SVM 0.596) trails FTLM 0.752 by $+0.156$ (t=11.62, $p<10^{-5}$). By dataset, FTLM's median delta spans $+0.143$ (EAModelSet Layer), $+0.309$ (ModelSet Abstract), and $+0.231$ (OntoUML Stereotype), with maxima of $+0.472$, $+0.895$, and $+0.564$, respectively.

RQ3 Conclusion. Finetuned pretrained transformers decisively outperform classical pipelines and transformer models trained from scratch on the same data. Pretraining provides essential semantic priors for modeling language text, making finetuning the practical and effective route, while from-scratch training and shallow baselines remain markedly behind.

VII. Discussion

In the following, we discuss the results from our empirical study.

A. Evidence from Empirical Results

Our extensive evaluation across four core tasks—node classification, edge classification, link prediction, and graph classification—demonstrates that GLM4MDE not only overcomes the limitations summarized in Table XI, but also establishes new state-of-the-art baselines on all benchmarks. Below we highlight the key findings and their broader implications.

Node Classification: When using only the fine-tuned LLM embeddings (i.e. without any GNN), we observe F_1 scores in the range 0.75–0.90 across five diverse classification problems. Incorporating GNN message passing elevates these scores in four of five datasets and provides comparative performance for the remaining one. Compared to classical baselines, our approach provided a consistent gain of +0.15–0.20 over both classical baselines (TF–IDF, GloVe, Worde4MDE + SVM/RF) and our in-house transformer (CMGPT) trained from scratch.
Edge Classification: Edge-type prediction is inherently fine-grained, requiring disambiguation among dozens of predicate

TABLE XI
SUMMARY OF IDENTIFIED LIMITATIONS, GLM4MDE RESOLUTIONS, AND EMPIRICAL EVIDENCE

Limitation	Description	How GLM4MDE Resolves It	Empirical Evidence
CL_1	NL semantics not utilized: bag-of-words/static embeddings ignore context and word-order.	Contextualized LLM embeddings fine-tuned on modeling language corpora capture word-order and long-range dependencies among labels, attributes, and types.	Fine-tuned LLM embeddings alone yield 0.75–0.90 F_1 on node classification, providing a +0.15–0.20 boost over `word2vec/SVM` baselines (RQ1).
CL_2	Modeling language semantics not utilized: Modeling Language primitives (e.g. "composition") are omitted.	Configurable MLS ingestion (parameters P1–P6) lets users include node-types and edge-types in the text (N_T/E_T).	Including edge-type semantics yields up to +0.03–0.10 F_1 improvement in node and edge classification vs. label-only (Sec. V-RQ2).
CL_3	Graph structure not integrated with NL semantics: text and structure pipelines remain separate.	Hybrid pipeline: LLM embeddings feed into GNN layers that learn to weight and propagate information according to topology.	`LM+GNN` achieves +0.04–0.10 increase over pure-LLM or pure-GNN baselines.
CL_4	Limited to a single modeling language: prior work targets one modeling language, lacking generality.	Language-agnostic CKG transformation: a metamodel parses any labeled, attributed graph.	Validated on three modeling languages, namely, ArchiMate, Ecore, and OntoUML, with extensibility support for other domain-specific modeling languages.
TL_1	Pretrained LLMs not finetuned: off-the-shelf models lack domain sensitivity.	Supported LLM finetuning allows any Huggingface transformer to adapt to model text.	Finetuned LLM outperforms CMGPT (trained from scratch) by +0.12–0.25 F_1 on node classification.
TL_2	OOV problem: static embeddings cannot represent unseen modeling jargon.	Subword tokenization (Huggingface) plus domain finetuning ensures every term is represented and prevents OOV failures.	Static `GloVe+SVM` scores only 0.353–0.410 F_1 on EAModelSet, whereas finetuned LLM achieves up to 0.843 F_1 (Table X).
TL_3	Rigid data configurations: feature extraction is hard-coded, limiting control.	Configurable text extraction exposes parameters (use_attributes, use_node_types, k-hop) for user-defined pipelines (Sec. IV-D).	Semantic ablation shows a +0.05 F_1 increase in edge classification when moving from $L_3 \rightarrow L_5$, demonstrating explainable feature impact.

labels. Here, the hybrid LM+GNN pipeline (particularly configurations using E_L and E_T) attains a near-perfect F_1 of 0.98–0.99 on both Ecore-555 and ModelSet, outperforming pure-LLM or pure-GNN systems. The magnitude of this improvement illustrates that i) contextualized token embeddings alone cannot capture the subtle distinctions in edge semantics, and b) the GNN's learnable propagation usefully weighs those distinctions according to the local topology.

Link Prediction: In link prediction, appending just one-hop neighbor text to the LLM inputs yields 0.83–0.95 F_1, matching or exceeding traditional graph-kernel methods. This result reveals that the textual co-occurrence of node labels can act as an efficient proxy for link prediction, beyond one hop, performance plateaus or declines—further evidence that message-passing layers are better suited than raw text to capture higher-order connectivity. Interestingly, while link prediction would intuitively seem like a structural task more than semantic, GNNs provide a poor performance for link prediction. It remains a topic of further research and investigation to improve and adapt GNNs for link prediction in this context.

Graph Classification: For assigning global domain labels to entire graphs, our approach reaches 0.90–0.96 F_1, outperforming both text-only and GNN-only alternatives in the case of Ecore and providing a lower but similar trend in performance for ModelSet. This confirms that combining semantic and structural modalities for an entire graph can also yield robust representations for models as graphs.

Key Configuration Insights: Three consistent patterns emerged from our parameter sweep:

1) *One-hop sufficiency.* The largest marginal gain occurs when k increases from 0 to 1 and sometimes from 1 to 2; beyond that, additional raw text contributes redundancy rather than novel information.
2) *Performance Variance in LLM versus GNN.* We see in Fig. 14 that there is more variance in the performance for different configurations while using LLMs, whereas with GNNs, the trend for different configurations is quite similar with overlapping lines. This indicates that LLMs are more sensitive to different configurations compared to GNNs.
3) *Selective semantics.* Including modeling language primitives (edge types, node types) delivers up to +0.05 F_1 on edge classification ($N_t \rightarrow E_T$), whereas link prediction is largely agnostic to these extras. This task-specific sensitivity highlights the value of our configurable extraction pipeline.

Explainability and Interpretability: Although pretrained LLMs and GNNs are often criticized as "black-box" models, GLM4MDE's modular design allows transparency. Our approach, by design, provides feature-ablation studies by allowing a practitioner to toggle parameters such as `use_node_types` or `k_hop`, such that she can measure the isolated impact of each semantic element, as reflected in our +0.03–0.10 F_1 gains. Together, these mechanisms empower modelers to audit and trust GLM4MDE's recommendations, mitigating concerns about opaque inference.

Value of Pretraining: Finally, our comparison of FTLM against CMGPT confirms that pretrained-then-finetuned models outperform classical ML and from-scratch transformers by +0.20–0.30 and +0.12–0.25 F_1 respectively. These results decisively demonstrate that the rich semantic priors acquired during large-scale language pretraining are irreplaceable when working with models having natural language semantics that need further finetuning.

By synthesizing these findings, we conclude that GLM4MDE resolves the conceptual and technical limitations of prior ML4MDE frameworks and provides a principled, transparent, and high-performance solution for a wide range of modeling tasks.

B. Integrating Generative AI and Predictive AI

As we elaborated, our work involved a classification-based partial model completion (PMC) approach. However, our approach can be synergistically integrated with the existing generative LLM-based conceptual modeling techniques to build a richer, more capable modeling assistant. Following a "Generate-then-Classify" approach where a modeler leaves placeholders (e.g., unnamed classes or relationships), invoke an LLM (e.g., GPT-4) in a generative mode to propose natural-language labels grounded in the partial model's context. This can be followed by feeding the LLM's suggested labels into GLM4MDE to assign the precise modeling language constructs (e.g., UML generalization vs. composition, ArchiMate node types) and ontological categories (e.g., kind, role). This approach cleanly separates "what to call it?" (generative) from "what exactly is it?" (classification). Furthermore, using the confidence scores of the classification model can be beneficial, such that if the confidence is low, it automatically falls back to asking the user for clarification or more context. Such an integrated approach can lead to domain-relevant names and guarantees structurally valid and ontologically rigorous modeling constructs. Such a unified framework can accelerate model construction, improve model quality, and foster trust by making each step transparent and controllable.

C. Threats to Validity

Our work is not free from threats to its validity. Below, we list the threats along the standard dimensions of threats to validity.

1) Internal Validity: Data leakage could occur if masked attributes inadvertently appear in training text. We mitigated these threats using identical train/test splits across all comparators, randomized seeds for each run, and strictly masked test-only attributes during text extraction. In several works, data leakage occurs by design due to human errors, which provides misleading results. Currently, our framework does not support automated indication of data leakage. All the experiments required careful manual inspection and rules to avoid data leakage (e.g., avoiding edge type information for the abstract class node classification task). Therefore, there is a need for a data leakage validator to ensure that the training data does not include information directly correlated to the final class that the classifier needs to predict. We aim to work on such a validator in our future work.
2) Construct Validity: We operationalize "capturing NL and modeling-language semantics" solely via closed-set classification F_1-scores on predefined node/edge categories. However, accuracy on these discrete labels may not truly reflect the system's semantic understanding—for instance, its ability to disambiguate homonyms, infer implicit relationships, or preserve nuanced domain concepts in downstream tasks (e.g., model merging or inconsistency resolution). High F_1 can mask systematic confusions among semantically adjacent classes (e.g., "Entity" vs. "Boundary") and overlook the quality of the learned embeddings beyond label prediction. We aim to mitigate this issue in future work, where we aim to complement quantitative metrics with intrinsic evaluations (e.g., probing tasks, semantic-similarity benchmarks) and qualitative user studies in which practitioners assess whether the model's suggestions and embedding proximities align with their mental model of the DSL.
3) External Validity: We evaluate on three diverse modeling languages, yet there exist many other DSLs (e.g., BPMN, SysML) with different syntactic constraints or multi-view semantics. Additionally, all datasets derive from open repositories and may not reflect proprietary industrial models. We mitigated this threat by implementing a modeling language agnostic layer, i.e., our CKG transformation, and we have released the framework's extensible parser—future users can readily plug in other DSLs. We also plan industrial case studies to assess generalizability.
4) Conclusion Validity: Statistical significance might be affected by multiple comparisons across k, configurations, and tasks; over-interpretation of small F_1 gains (¡0.01) may be misleading. We report average results over five runs, include confidence intervals (±0.005–0.015 F_1), and highlight only robust improvements ($\geq$0.02 F_1) as meaningful.

While no empirical study can eliminate all threats, our careful design, open release of code and data, and use of multiple tasks and languages bolster confidence that GLM4MDE's advantages are both real and broadly applicable.

VIII. Conclusion and Future Work

This paper presented GLM4MDE, a Graph Language Modeling framework that unifies pretrained large language models and graph neural networks to learn richly contextualized representations of conceptual models. We filled critical gaps in prior ML4MDE approaches by integrating NL semantics, modeling language primitives, and graph topology in a single, configurable pipeline. We then demonstrated state-of-the-art performance on four datasets spanning three different modeling languages and four core tasks (node/edge classification, link prediction, graph classification), with F_1 gains of +0.15–0.45 over classical baselines and scratch models. We validated the necessity of pretrained LLMs and the complementarity of the

static textual context of nodes, embedded with finetuned large language models and then trained with GNN message-passing, showing the combined power of LLMs with GNNs. Finally, we delivered an open-source Python package empowering model engineers to harness advanced ML for domains beyond those studied here[16].

By combining transformers' lexical power with graph networks' structural rigor, GLM4MDE paves the way toward smarter, more automated modeling tools that can assist practitioners across heterogeneous DSLs. In future work, we aim to explore full PMC pipelines combining generative LLMs with our classification backbone, scaling to industrial-sized models, and embedding our framework into visual modeling environments for live assistance.

REFERENCES

[1] H. A. Proper and G. Guizzardi, "Modeling for enterprises; let's go to rome via rime," *hand*, vol. 1, p. 3, 2022.

[2] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis lectures on software engineering*, vol. 3, no. 1, pp. 1–207, 2017.

[3] J. Michael, D. Bork, M. Wimmer, and H. C. Mayr, "Quo vadis modeling?" *Software and Systems Modeling*, pp. 1–22, 2023.

[4] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[5] A. Jacobsen, R. de Miranda Azevedo, N. Juty, D. Batista, S. Coles, R. Cornet, M. Courtot, M. Crosas, M. Dumontier, C. T. Evelo *et al.*, "Fair principles: interpretations and implementation considerations," *Data Intelligence*, vol. 2, no. 1-2, pp. 10–29, 2020.

[6] J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Collaborative repositories in model-driven engineering [software technology]," *IEEE Software*, vol. 32, no. 3, pp. 28–34, 2015.

[7] A. C. Marcén, A. Iglesias, R. Lapeña, F. Pérez, and C. Cetina, "A systematic literature review of model-driven engineering using machine learning," *IEEE Transactions on Software Engineering*, 2024.

[8] M. B. Chaaben, L. Burgueño, I. David, and H. Sahraoui, "On the utility of domain modeling assistance with large language models," *arXiv preprint arXiv:2410.12577*, 2024.

[9] M. Weyssow, H. Sahraoui, and E. Syriani, "Recommending metamodel concepts during modeling activities with pre-trained language models," *Software and Systems Modeling*, vol. 21, no. 3, pp. 1071–1089, 2022.

[10] S. J. Ali and D. Bork, "A graph language modeling framework for the ontological enrichment of conceptual models," in *International Conference on Advanced Information Systems Engineering*. Springer, 2024, pp. 107–123.

[11] C. Arora, M. Sabetzadeh, S. Nejati, and L. Briand, "An active learning approach for improving the accuracy of automated domain model extraction," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 1, pp. 1–34, 2019.

[12] R. Saini, G. Mussbacher, J. L. Guo, and J. Kienzle, "Domobot: A modelling bot for automated and traceable domain modelling," in *2021 IEEE 29th International Requirements Engineering Conference (RE)*. IEEE, 2021, pp. 428–429.

[13] L. Burgue
textasciitilde no, J. Cabot, and S. Gérard, "An lstm-based neural network architecture for model transformations," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2019, pp. 294–299.

[14] K. Lano, S. Kolahdouz-Rahimi, and S. Fang, "Model transformation development using automated requirements analysis, metamodel matching, and transformation by example," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–71, 2021.

[15] P. T. Nguyen, D. Di Ruscio, A. Pierantonio, J. Di Rocco, and L. Iovino, "Convolutional neural networks for enhanced classification mechanisms of metamodels," *Journal of Systems and Software*, vol. 172, p. 110860, 2021.

[16]https://pypi.org/project/glam4cm/

[16] J. A. H. López, J. S. Cuadrado, R. Rubei, and D. Di Ruscio, "Modelxglue: a benchmarking framework for ml tools in mde," *Software and Systems Modeling*, pp. 1–24, 2024.

[17] G. Guizzardi, A. Botti Benevides, C. M. Fonseca, D. Porello, J. P. A. Almeida, and T. Prince Sales, "Ufo: Unified foundational ontology," *Applied ontology*, vol. 17, no. 1, pp. 167–210, 2022.

[18] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations. arxiv 2018," *arXiv preprint arXiv:1802.05365*, vol. 12, 2018.

[19] I. Tenney, P. Xia, B. Chen, A. Wang, A. Poliak, R. T. McCoy, N. Kim, B. Van Durme, S. R. Bowman, D. Das *et al.*, "What do you learn from context? probing for sentence structure in contextualized word representations," *arXiv preprint arXiv:1905.06316*, 2019.

[20] M. Polignano, C. Musto, M. de Gemmis, P. Lops, and G. Semeraro, "Together is better: Hybrid recommendations combining graph embeddings and contextualized word representations," in *Proceedings of the 15th ACM conference on recommender systems*, 2021, pp. 187–198.

[21] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," *arXiv preprint arXiv:1903.10676*, 2019.

[22] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "Biobert: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.

[23] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[24] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[25] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM computing surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[26] P.-L. Glaser, E. Sallinger, and D. Bork, "Ea modelset–a fair dataset for machine learning in enterprise modeling," in *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer, 2023, pp. 19–36.

[27] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, and R. Rubei, "On the use of large language models in model-driven engineering," *Software and Systems Modeling*, pp. 1–26, 2025.

[28] N. Mu, S. Chen, Z. Wang, S. Chen, D. Karamardian, L. Aljeraisy, B. Alomair, D. Hendrycks, and D. Wagner, "Can llms follow simple rules?" *arXiv preprint arXiv:2311.04235*, 2023.

[29] D. Bajaj, A. Goel, S. Gupta, and H. Batra, "Muce: a multilingual use case model extractor using gpt-3," *International Journal of Information Technology*, vol. 14, no. 3, pp. 1543–1554, 2022.

[30] Y. Yang, B. Chen, K. Chen, G. Mussbacher, and D. Varró, "Multi-step iterative automated domain modeling with large language models," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, 2024, pp. 587–595.

[31] B. Chen, K. Chen, S. Hassani, Y. Yang, D. Amyot, L. Lessard, G. Mussbacher, M. Sabetzadeh, and D. Varró, "On the use of gpt-4 for creating goal models: an exploratory study," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, 2023, pp. 262–271.

[32] H.-G. Fill, P. Fettke, and J. Köpke, "Conceptual modeling and large language models: impressions from first experiments with chatgpt," *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, vol. 18, pp. 1–15, 2023.

[33] Y. Xia, J. Kim, Y. Chen, H. Ye, S. Kundu, C. C. Hao, and N. Talati, "Understanding the performance and estimating the cost of llm fine-tuning," in *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2024, pp. 210–223.

[34] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen *et al.*, "Siren's song in the ai ocean: a survey on hallucination in large language models," *arXiv preprint arXiv:2309.01219*, 2023.

[35] K. Chen, Y. Yang, B. Chen, J. A. H. López, G. Mussbacher, and D. Varró, "Automated domain modeling with large language models: A comparative study," in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2023, pp. 162–172.

[36] J. A. H. López, C. Durá, and J. S. Cuadrado, "Experimenting with modeling-specific word embeddings," *Software and Systems Modeling*, pp. 1–23, 2024.

[37] K. P. Sinaga and M.-S. Yang, "Unsupervised k-means clustering algorithm," *IEEE access*, vol. 8, pp. 80 716–80 727, 2020.

[38] J. Sequeda and O. Lassila, "Designing and building enterprise knowledge graphs," *Synthesis Lectures on Data, Semantics, and Knowledge*, vol. 11, no. 1, pp. 1–165, 2021.

[39] M. Smajevic and D. Bork, "Towards graph-based analysis of enterprise architecture models," in *International Conference on Conceptual Modeling*. Springer, 2021, pp. 199–209.

[40] S. Sun, F. Meng, and D. Chu, "A model driven approach to constructing knowledge graph from relational database," in *Journal of Physics: Conference Series*, vol. 1584, no. 1. IOP Publishing, 2020, p. 012073.

[41] S. J. Ali, G. Guizzardi, and D. Bork, "Enabling representation learning in ontology-driven conceptual modeling using graph neural networks," in *CAiSE'23*, 2023.

[42] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002.

[43] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of EMNLP*, 2013, pp. 1631–1642.

[44] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[45] B. Warner, A. Chaffin, B. Clavié, O. Weller, O. Hallström, S. Taghadouini, A. Gallagher, R. Biswas, F. Ladhak, T. Aarsen *et al.*, "Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference," *arXiv preprint arXiv:2412.13663*, 2024.

[46] D. S. Nielsen, K. Enevoldsen, and P. Schneider-Kamp, "Encoder vs decoder: Comparative analysis of encoder and decoder language models on multilingual nlu tasks," *arXiv preprint arXiv:2406.13469*, 2024.

[47] M. Qorib, G. Moon, and H. T. Ng, "Are decoder-only language models better than encoder-only language models in understanding word meaning?" in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 16 339–16 347.

[48] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[49] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[50] D. Strüber, M. Selter, and G. Taentzer, "Tool support for clustering large meta-models," in *Proceedings of the workshop on scalability in model driven engineering*, 2013, pp. 1–4.

[51] A. Elkamel, M. Gzara, and H. Ben-Abdallah, "An uml class recommender system for software design," in *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*. IEEE, 2016, pp. 1–8.

[52] J. J. Jiang and D. W. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," *arXiv preprint cmp-lg/9709008*, 1997.

[53] F. Basciani, J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Automated clustering of metamodel repositories," in *Advanced Information Systems Engineering: 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings 28*. Springer, 2016, pp. 342–358.

[54] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[55] L. Addazi, A. Cicchetti, J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Semantic-based model matching with emfcompare." in *Me@ models*, 2016, pp. 40–49.

[56] Ö. Babur, L. Cleophas, and M. van den Brand, "Hierarchical clustering of metamodels for comparative analysis and visualization," in *European conference on modelling foundations and applications*. Springer, 2016, pp. 3–18.

[57] Ö. Babur and L. Cleophas, "Using n-grams for the automated clustering of structural models," in *International Conference on Current Trends in Theory and Practice of Informatics*. Springer, 2017, pp. 510–524.

[58] R. Rubei, J. Di Rocco, D. Di Ruscio, P. T. Nguyen, and A. Pierantonio, "A lightweight approach for the automated classification and clustering of metamodels," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2021, pp. 477–482.

[59] P. T. Nguyen, J. Di Rocco, L. Iovino, D. Di Ruscio, and A. Pierantonio, "Evaluation of a machine learning classifier for metamodels," *Software and Systems Modeling*, vol. 20, no. 6, pp. 1797–1821, 2021.

[60] A. Khalilipour, F. Bozyigit, C. Utku, and M. Challenger, "Categorization of the models based on structural information extraction and machine learning," in *International Conference on Intelligent and Fuzzy Systems*. Springer, 2022, pp. 173–181.

[61] U. Kang, H. Tong, and J. Sun, "Fast random walk graph kernel," in *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 2012, pp. 828–838.

[62] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[63] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.

[64] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[65] J. A. H. López, R. Rubei, J. S. Cuadrado, and D. Di Ruscio, "Machine learning methods for model classification: a comparative study," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, 2022, pp. 165–175.

[66] L. Burgueño, R. Clarisó, S. Gérard, S. Li, and J. Cabot, "An nlp-based architecture for the autocompletion of partial domain models," in *International Conference on Advanced Information Systems Engineering*. Springer, 2021, pp. 91–106.

[67] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, and A. Pierantonio, "Memorec: a recommender system for assisting modelers in specifying metamodels," *Software and Systems Modeling*, vol. 22, no. 1, pp. 203–223, 2023.

[68] C. Di Sipio, J. Di Rocco, D. Di Ruscio, and P. T. Nguyen, "Morgan: a modeling recommender system based on graph kernel," *Software and Systems Modeling*, vol. 22, no. 5, pp. 1427–1449, 2023.

[69] J. Silva, Q. Ma, J. Cabot, P. Kelsen, and H. A. Proper, "Application of the tree-of-thoughts framework to llm-enabled domain modeling," in *Conceptual Modeling - 43rd International Conference, ER 2024, Pittsburgh, PA, USA, October 28-31, 2024, Proceedings*, ser. Lecture Notes in Computer Science, W. Maass, H. Han, H. Yasar, and N. J. Multari, Eds., vol. 15238. Springer, 2024, pp. 94–111.

[70] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[71] Djelic, S. J. Ali, C. Verbruggen, J. Neidhardt, and D. Bork, "A model cleansing pipeline for model-driven engineering: Mitigating the garbage in, garbage out problem for open model repositories," in *2025 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*, To Appear. [Online]. Available: https://model-engineering.info/publications/papers/MODELS25-ModelCleansing.pdf

[72] H. Zhao, H. Chen, F. Yang, N. Liu, H. Deng, H. Cai, S. Wang, D. Yin, and M. Du, "Explainability for large language models: A survey," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 2, pp. 1–38, 2024.

[73] J. A. H. López, J. L. Cánovas Izquierdo, and J. S. Cuadrado, "Modelset: a dataset for machine learning in model-driven engineering," *Software and Systems Modeling*, pp. 1–20, 2022.

[74] T. P. Sales, P. P. F. Barcelos, C. M. Fonseca, I. V. Souza, E. Romanenko, C. H. Bernabé, L. O. B. da Silva Santos, M. Fumagalli, J. Kritz, J. P. A. Almeida *et al.*, "A fair catalog of ontology-driven conceptual models," *Data & Knowledge Engineering*, vol. 147, p. 102210, 2023.

[75] B. Rosner, R. J. Glynn, and M.-L. T. Lee, "The wilcoxon signed rank test for paired comparisons of clustered data," *Biometrics*, vol. 62, no. 1, pp. 185–192, 2006.

[76] J. A. H. López, C. Durá, and J. S. Cuadrado, "Word embeddings for model-driven engineering," in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2023, pp. 151–161.